Numer Algor (2008) 47:341–359
DOI 10.1007/s11075-008-9192-9

ORIGINAL PAPER

Block Krylov–Schur method for large symmetric eigenvalue problems

Yunkai Zhou · Yousef Saad

Received: 24 January 2008 / Accepted: 26 February 2008 /
Published online: 27 March 2008

Abstract Stewart's Krylov–Schur algorithm offers two advantages over Sorensen's implicitly restarted Arnoldi (IRA) algorithm. The first is ease of deflation of converged Ritz vectors, the second is the avoidance of the potential forward instability of the QR algorithm. In this paper we develop a block version of the Krylov–Schur algorithm for symmetric eigenproblems. Details of this block algorithm are discussed, including how to handle rank deficient cases and how to use varying block sizes. Numerical results on the efficiency of the block Krylov–Schur method are reported.

Keywords Block method · Krylov–Schur · Lanczos · Implicit restart

1 Introduction

Sorensen's implicitly restarted Arnoldi (IRA) algorithm [27] is one of the most successful eigenvalue methods and the ARPACK package [17] based on IRA is the best-known public package and the workhorse for large eigenproblems ever since its appearance in the late 1990s. In [32] Stewart proposed the Krylov–Schur method which is mathematically equivalent to

Work supported by US Department of Energy under contract DE-FG02-03ER25585, by NSF grants ITR-0428774 and CMMI-0727194, and by the Minnesota Supercomputing Institute.

Y. Zhou (✉)
Department of Mathematics, Southern Methodist University,
Dallas, TX 75275, USA
e-mail: yzhou@smu.edu

Y. Saad
Department of Computer Science and Engineering, University of Minnesota,
Minneapolis, MN 55455, USA
e-mail: saad@cs.umn.edu

IRA but offers two practical advantages. First, it is easier to deflate converged Ritz vectors. Second, the potential forward instability of the QR algorithm [20, 34] is avoided. This forward instability can cause unwanted Ritz vectors to persist in the Arnoldi decomposition, hence one needs to employ additional elaborate purging procedures to purge the unwanted Ritz vectors [16, 28]. These two advantages are gained by giving up the strict upper Hessenberg form in the Arnoldi decomposition. Instead a Rayleigh quotient matrix is used, which leads to the (general) Krylov decomposition, as proposed in [32]. For numerical consideration we restrict ourselves to the orthonormal Krylov decomposition which may be considered as a general Arnoldi factorization. One performs the Schur decomposition on the Rayleigh quotient matrix. Note that the Ritz values are contained in the diagonal blocks of the (quasi)-triangular Schur factor, therefore, it is easier to perform deflation and purging with this decomposition than with the Arnoldi decomposition, which uses Rayleigh quotient of upper Hessenberg form. The advantage is more obvious for symmetric eigenvalue problems because the (quasi)-triangular Schur factor becomes diagonal. In the symmetric case, the Krylov–Schur method is called *Krylov-spectral* [33], since the Schur decomposition reduces to the spectral decomposition. As mentioned in [32], the *Krylov-spectral* method is identical to the thick restart method of Wu and Simon [35]. Here we note that the advantages of *Krylov-spectral* are gained by losing one advantage of IRA—the flexibility in applying shifts that are not Ritz values.

In this paper we develop a block version of the Krylov–Schur method for symmetric eigenproblems. We notice a third advantage of Krylov–Schur method over IRA: It is relatively easier to develop a block version method based on the Krylov–Schur structure. The reason is that in the contraction phase, the Krylov–Schur method does not use implicit shifted QR to filter unwanted Ritz values; while a genuine block version IRA would require block implicit shifted QR decompositions to apply unwanted Ritz values as shifts. Efficient methods for block implicit shifted QR decomposition have been hard to construct (e.g. [18]). The IRBL method presented in [1, 2] actually uses explicit shifted QR decomposition for this purpose, which may not be as stable as the original IRA/IRL based on implicit QR decomposition.

The need for a block version eigensolver arises in many applications. Block eigensolvers remain important throughout the development of modern numerical linear algebra. Literature on block eigensolvers includes [1, 3, 6, 8–12, 21, 23–26]. Both block methods and non-block methods have their merits (see [19, pp. 316–320] for a concise discussion). Block methods are more efficient for multiple or clustered eigenvalues. Moreover, a block method is the natural choice when several good initial vectors are available. This situation is common for the self-consistency iteration in electronic structure calculations, where a former loop provides good initial vectors for the next loop. One other advantage of a block method over a non-block method is better utilization of cache. This can yield a significant gain for large dense matrix–vector products; while for large sparse matrix–vector products, the cache performance gain comes from less frequent access to the storage scheme (data structure) describing the sparse matrix; although this gain may be less significant if the matrix–vector products can be directly coded without using any storage schemes.

The advantages of block methods come with extra complexities in algorithm description and coding. The most noticeable complexity is the so called *rank deficient case*, which is the situation when some vectors in a new block become linearly dependent. Even though much was written on block methods, there lack detailed discussions on the rank deficient cases. It is likely that different authors have different implementations. A block Lanczos method was proposed in [8] with a detailed discussion of deflation of linearly dependent Lanczos vectors, but the method can only use BLAS-2 and the block size keeps decreasing. In this paper we give a detailed discussion on how to deal with rank deficiency. Rank revealing pivoted QR decomposition (BLAS-3) is used for the rank deficient case. Block size is reduced whenever there is rank deficiency. In the block Krylov–Schur method, the block size can also be increased naturally, this is discussed in Section 2.2. Therefore the block size is adapted when necessary. Adaptive block Lanczos methods exist in [3, 36], where block size is increased when needed. In contrast, the block size in our block Lanczos code is not increased, block size is increased only in the Krylov–Schur cycle if necessary.

In the next section we describe the block Krylov–Schur method for symmetric eigenproblems. We continue to use the term *Krylov–Schur* instead of *Krylov-spectral* to follow the terminology initiated by Stewart. We present a detailed discussion on the rank deficient case in Section 2.1. The idea of adaptive block size is discussed in Section 2.2. Note that the adaptive idea in [3] only refers to increasing block size for the Lanczos decomposition, while here we refer to both increasing and decreasing block size adaptively inside the Krylov–Schur loop. In Section 3 we discuss the block Lanczos decomposition, which is an important step for the Krylov–Schur method. Section 5 presents numerical results of block Krylov–Schur, including comparisons of our Matlab code to the Matlab codes IRBL [1, 2], LOBPCG [11] and Matlab *eigs*, and comparisons of our Fortran code to ARPACK [17].

2 Block Krylov–Schur for symmetric eigenproblems

We now describe the Krylov–Schur method. The Krylov–Schur method belongs to the implicit restart category, i.e., the restarting vector is obtained not by explicit polynomial filtering but by implicit filtering. Sorensen [27] achieved the implicit polynomial filtering by utilizing the property of the shifted QR algorithm. The implicit polynomial filtering for Krylov–Schur is not obvious. However, the following theorem shows the equivalence between Krylov–Schur and IRA. We denote by $IRA(k, m)$ and $KS(k, m)$, the subspaces with maximum dimension m which is contracted to dimension k at restart, by the IRA and Krylov–Schur method, respectively.

Theorem 2.1 *Starting from the same initial vector, if at each restart, $KS(k, m)$ and $IRA(k, m)$ filter away the same $m-k$ Ritz values that are distinct from the remaining k Ritz values, then the basis vectors after contraction of both methods span the same subspace.*

Proof See [32] or [33, pp. 331–332]. □

Theorem 3.4 in [31] contains essentially the same equivalence result as the above Theorem 2.1 but with a different proof. Theorem 2.1 holds true for a general matrix, hence it is also true for the symmetric case that we are interested in.

The block Krylov–Schur method is a natural block extension of the Krylov–Schur method. From Theorem 2.1 we know that the single vector Krylov–Schur method is mathematically equivalent to the highly successful IRA. As mentioned in the introduction, the Krylov–Schur method has two advantages over IRA. Therefore we can expect the block Krylov–Schur method to offer the efficiency of IRA, the ease of deflation, and the advantages of a block method.

The general cycle of block Krylov–Schur method for symmetric $\mathcal{H} \in \mathbb{R}^{n\times n}$ contains four steps: (here b denotes the block size; k_s denotes the starting basis size, it is also the basis size after contraction; k_f denotes the final basis size; $k_s < k_f$; and V denotes the orthonormal basis, T the Rayleigh-quotient matrix)

1. Augment a size k_s block Krylov decomposition, $\mathcal{H}V_{k_s} = V_{k_s}T_{k_s} + FB^T$, where $F \perp V_{k_s}$, $F \in \mathbb{R}^{n\times b}$, $B \in \mathbb{R}^{k_s\times b}$, to a size k_f block Krylov decomposition: $\mathcal{H}V_{k_f} = V_{k_f}T_{k_f} + \tilde{F}E_{k_f}^T$, $(\tilde{F} \perp V_{k_f})$;
   (this is essentially a block Lanczos augmentation step);
2. Compute the Schur (spectral) decomposition of T_{k_f}:
   $T_{k_f}Q_{k_f} = Q_{k_f}D_{k_f}$, where the Ritz values are ordered so that the first k_s Ritz pairs are wanted pairs;
3. Contract the size k_f orthogonal basis to size k_s:
   $V_{k_s} \leftarrow V_{k_f}Q_{k_f}(:, 1 : k_s)$;
4. The size k_s Krylov decomposition now can be written as:
   $\mathcal{H}V_{k_s} = V_{k_s}D_{k_s} + \tilde{F}B^T$, where $B^T = E_{k_f}^T Q_{k_f}(:, 1 : k_s)$;
   repeat from step 1.

Actually, at the very beginning of the block Krylov–Schur cycle, one applies a block Lanczos iteration to build a size k_f Lanczos decomposition. The method then iterates $2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ as listed above. The multiplication by $\mathcal{H}$ at step 1 is usually achieved via a user supplied external subroutine.

The augmentation of Krylov decomposition by block Lanczos at step 1 is important, and will be discussed in Section 3. We devote a subroutine `block_lanczos()` (presented later in Algorithm 3.1) for this augmentation. The variable *blkcontrol* is used to control the block size. With this block Lanczos decomposition ready, it is relatively easy to describe the block Krylov–Schur method in a more detailed pseudo code. We first call `block_lanczos()` to get a size-k_f Lanczos decomposition, then compute the spectral decomposition of the size-k_f Rayleigh-quotient matrix T. Then the size-k_f basis V is contracted into a size-k_s basis, based on the selection criteria of the computed Rayleigh–Ritz pairs.

In order to directly call the `block_lanczos()` code without changing the interface, in the block Krylov–Schur code we add one block of basis vectors to V and update T accordingly before calling `block_lanczos()` again. As seen from the first and third pictures in Fig. 1, after the contraction and after adding the residual terms to the last rows of T, we do one more step of block augmentation, which adds the last block_size columns to T and makes T symmetric before passing it to `block_lanczos()` for further augmentation.

The detailed pseudo code is shown in Algorithm 2.1 (Fig. 2). Where the integer *max_subspdim* is the maximum subspace dimension allowed; and *blkcontrol* = 1 means that the block size can be increased when necessary. The seemingly complicated indexing in the pseudo code handles the deflation of converged Ritz pairs. In the next subsections we provide more details on Algorithm 2.1. One important detail of the block Krylov–Schur method is how to handle the rank deficient case.

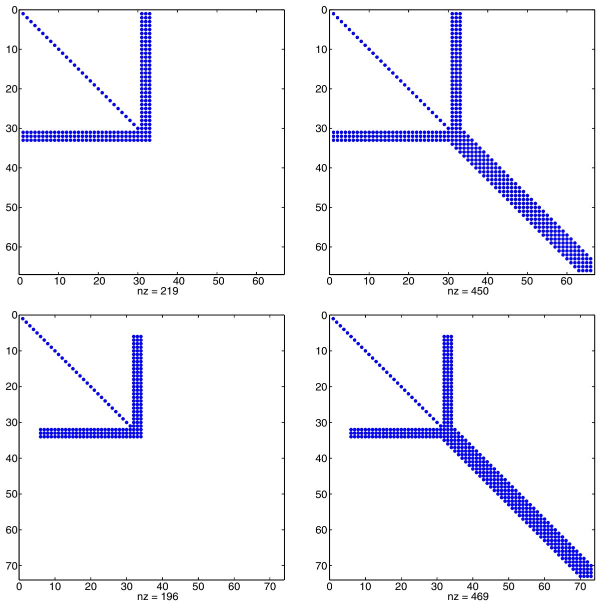


Fig. 1 Sample structure of the Rayleigh-quotient matrix T (block_size=3). The *first figure* contains the diagonal term after the contraction phase, with residual terms added at the last block_size rows. One additional step block augmentation leads to the last block_size columns which makes T symmetric. The *second figure* is the contracted T augmented by block Lanczos. The *third figure* also contains the diagonal term after the contraction phase with correction terms added. It is the same as the first figure, except that the converged Ritz values at the beginning of the diagonal have been deflated, so the active T only consists of the part with nonzero residuals (or off-diagonals). The *fourth figure* is the deflated T augmented by block Lanczos. Note that the size of the active T remains the same because the size of T increases after each deflation

ALGORITHM 2.1.

```
Input Data: $F, b, k_s, k_f$, blkcontrol, nneed, max_restart, max_subspdim, tol
Output Data: $V, T$, nconv
 I.  Apply block Lanczos iteration to get a size $k_f$ Lanczos decomposition: $\mathcal{H}V(:,1:k_f) = V(:,1:k_f)T(1:k_f,1:k_f) + FE^T$,
     s.t. $V(:,1:k_f)^T V(:,1:k_f) = I_{k_f}$,  $F^T V(:,1:k_f) = \mathbf{0}$;
     Set $nc = nconv = 0$;
II.  Do while (max_restart is not reached)
        1.  Compute the full spectral decomposition of $T(nc+1:k_f, nc+1:k_f)$: $T(nc+1:k_f, nc+1:k_f)X = XD$;
        2.  Keep only the first $k_s$ columns of $X$; contract the basis as
            $V(:,nc+1:nc+k_s) = V(:,nc+1:nc+k_f)X(:,1:k_s)$;
        3.  Compute rank revealing pivoted thin QR decomposition of $F$:
            $F(:,\mathbf{e}) = V(:,nc+k_s+1:nc+k_s+b)R$; set $rankF = rank(F)$;
        4.  Compute $\mathbf{inve}$ s.t. $\mathbf{e}(\mathbf{inve}(j)) = j$, for $j = 1:b$;
        5.  Compute the residuals $\hat{R} = R(:,\mathbf{inve})X(k_f-nc-b+1:k_f-nc,:)$;
            determine convergence from the residuals. Let $nc_0$ be the number of
            converged Ritz pairs in this step. If $nc_0 > 0$, set $nconv = nconv + nc_0$.
            If $nconv >= nneed$ then exit.
        6.  Update $T$ to include the non-zero residual:
            $T(nc+1:nc+k_s+b, nc+1:nc+k_s) = \begin{bmatrix} \hat{D} \\ \hat{R} \end{bmatrix}$
            where $\hat{D} = D(nc_0+1:nc_0+k_s, nc_0+1:nc_0+k_s)$;
        7.  If $(rankF < b)$ then ortho-normalize $V(:,nc+k_s+rankF+1:nc+k_s+b)$ against $V(:,1:nc+k_s+rankF)$;
        8.  If (blkcontrol==1), determine a new $b$ based on clustering size of
            diag($\hat{D}$); Augment $V$ and $T$ according to Eq. 7 if $b$ is increased;
        9.  Compute $F = \mathcal{H}V(:,nc+k_s+1:nc+k_s+b)$;
       10.  Compute $\mathbf{h} = V(:,1:nc+k_s+b)^T F$;
       11.  Get the diagonal block of $T$:
            $T(nc+k_s+1:nc+k_s+b, nc+k_s+1:nc+k_s+b) =$
            $\mathbf{h}(nc+k_s+1:nc+k_s+b,:)$;
       12.  Compute new residual vectors that are orthogonal to
            $V(:,1:nc+k_s)$:  $F = F - V(:,1:nc+k_s+b)\mathbf{h}$; Do selected
            re-orthogonalization s.t. $F \perp V(:,1:nc+k_s+b)$;
       13.  Set $k = nc+k_s+b$, $k_f = min(nc+k_f, max\_subspdim)$;
            set $nc = nconv$;
       14.  Call block_lanczos($V, T, F, k, k_f, b$, blkcontrol) to augment the
            size-k Krylov decomposition to size-$k_f$;
     End Do.
```

Fig. 2 Pseudocode for the Block Krylov–Schur method

2.1 Handling the rank deficient case

Implementations of block methods are more complicated than their non-block counterparts, partly because vectors in a new block can become linearly dependent at some stage of the iteration. This rank deficiency can appear in any block method. Hence what we discuss in this subsection also treats the rank deficient case for the block Lanczos decomposition in Section 3.

Suppose we already have a size-k block Krylov–Schur decomposition of $\mathcal{H} \in \mathbb{R}^{n\times n}$,

$$\begin{aligned} \mathcal{H}V_k &= V_k T_k + FB^T, \\ V_k^T V_k &= I_k, \quad F^T V_k = \mathbf{0}, \quad F \in \mathbb{R}^{n\times b}. \end{aligned} \tag{1}$$

The next step is to augment the size-k orthonormal basis V_k into a size-$(k+b)$ orthonormal basis. Let the QR decomposition of F be $F = QR$. Then (1) can be written as

$$\mathcal{H}V_k = [V_k, Q] \begin{bmatrix} T_k \\ RB^T \end{bmatrix}. \tag{2}$$

If F is of full rank, then we know $Q^T V_k = \mathbf{0}$ and Q contains the wanted size-b new augmentation vectors. In the case when F is rank deficient, we need a more careful examination of the QR decomposition of F. A rank revealing pivoted QR decomposition is more appropriate for this case. Let the pivoted thin QR decomposition of F be $FP = QR$, where P is the permutation matrix. We express QR in the following rank revealing form,

$$FP = QR = [Q_1, Q_2] \begin{bmatrix} [R_{11}, \ R_{12}] \\ \mathbf{0} \end{bmatrix} = Q_1[R_{11}, \ R_{12}], \tag{3}$$

where $Q_1 \in \mathbb{R}^{n\times r}$ and R_{11} is a $r \times r$ non-singular upper triangular matrix ($r := rank(R) = rank(F)$). From (3) it is clear that $F \perp V_k$ now only guarantees $Q_1 \perp V_k$, but not $Q_2 \perp V_k$.

For a block method with fixed block size, Q_2 must be re-orthonormalized against V_k to make sure that the augmented basis $[V, Q_1, Q_2]$ will continue to be orthonormal. Using Q_2 instead of other vectors avoids the need to generate new vectors and the need to orthogonalize the new vectors against Q_1. This is because the block Gram–Schmidt orthogonalization

$$Q_2 \leftarrow Q_2 - V_k(V_k^T Q_2),$$

with re-orthogonalization if necessary, gives a Q_2 that still satisfies $Q_2 \perp Q_1$. Certainly this favorable situation may be lost if some vectors in Q_2 happen to be in $range\{V_k\}$. In this case one has to generate random vectors to replace the zero vectors produced from the block Gram-Schmidt step, and one can not waive the re-orthogonalization of the newly generated vectors against Q_1.

In order to uniformly treat all possible cases, including the case when some vectors in Q_2 may be in $range\{V_k\}$, we choose to use not block Gram-Schmidt but instead a single vector version Gram-Schmidt with re-orthogonalization. This approach is essentially the DGKS [7] method that is used in ARPACK [17]. The difference is that this single vector version DGKS is now used in a block method. Our code orthonormalizes vectors in Q_2, one by one against V_k until all vectors in Q_2 that are not in $range\{V_k\}$ are used up. This process will produce $\hat{Q}_2 \perp [V_k, Q_1]$. During the process, the second step Gram-Schmidt refinement will count the number of vectors in Q_2 that are in $range\{V_k\}$, these vectors will be replaced by random vectors after all vectors in Q_2 are tried. And the random vectors are orthonormalized against $[V_k, Q_1, \hat{Q}_2]$ to give the desired size-$(k+b)$ orthonormal basis.

2.2 Using adaptive block sizes

In a block method, the block size need not be fixed. It is advantageous that the method can adaptively adjust its block size at different stages of the iteration. We propose two natural ways to achieve this goal. Note that in both cases the goal is to keep a valid $F = QM$ decomposition where Q is an orthogonal matrix which contains the orthonormal basis of F, and $M = Q^T F$. After this decomposition is achieved, the Krylov augmentation can be carried out in a standard way.

The first is to shrink the block size when F becomes rank deficient. That is, if the rank revealing QR of F is (3) and $0 < r < b$, then the block size b is reduced to r. The basis is augmented as $[V_k, Q_1]$ and the triangular factor R replaced by

$$[R_{11}, R_{12}]P^{-1}. \tag{4}$$

In the extremal case that $r = 0$, one can reset the block size to any suitable value. A good choice is to apply the method discussed above by orthogonalizing vectors in Q_2 against V_k, and set the new block size as the number of vectors in Q_2 that are not in $range\{V_k\}$. If this number is 0, one has to generate random vectors and perform a suitable orthogonalization to produce orthonormal vectors for augmentation. In any of these cases when $r = 0$, the update of the upper triangular factor R is simply a zero matrix of the same size as the updated block size.

The second is to adaptively increase the block size if the current block size is determined to be too small. This readily fits in the block Krylov–Schur method. Because in Algorithm 2.1 step III.1 we compute the eigenvalues of T, if the size of clustering of these Ritz values is larger than the current block size, then we can increase the block size to be at least the size of the clustering. This idea is in the same vein as in [3]. The difference is that in [3], the block size is adjusted at each step, which means the Ritz values have to be computed at each step of the block Lanczos iteration; while in block Krylov–Schur, we adjust the block size only after step III.1 of Algorithm 2.1 when the current Ritz values become available. The advantage is that we can keep the standard update of block Lanczos which does not require Ritz value computations at each step; moreover, adjusting block size too frequently can increase the coding complexity but may not offer much computational efficiency because of the extra Ritz value computations and block size determinations.

To increase the block size we make the following observations. Suppose that the pivoted QR decomposition of F [same notations as in (3)] is,

$$F = QRP^{-1} = [Q_1, Q_2]\begin{bmatrix} [R_{11}, \ R_{12}] \\ \mathbf{0} \end{bmatrix} P^{-1} = Q_1[R_{11}, \ R_{12}]P^{-1}, \tag{5}$$

where F can be of full rank or rank deficient (if $r = b$ then $R_{12} = \emptyset$). Then the size-k block Krylov decomposition $\mathcal{H}V_k = V_k T_k + FB^T$ becomes,

$$\begin{aligned}\mathcal{H}V_k &= V_k T_k + Q_1[R_{11}, R_{12}]P^{-1}B^T \\ &= [V_k, Q_1]\begin{bmatrix} T_k \\ [R_{11}, R_{12}]P^{-1}B^T \end{bmatrix}. \end{aligned} \tag{6}$$

Suppose now the block size should be increased to $\hat{b} > b$. One just needs to construct $\hat{b} - r$ new basis vectors. This is achieved by generating $\hat{b} - b$ random vectors, orthogonalizing these vectors and the $b - r$ vectors Q_2 in (5) against $[V_k, Q_1]$ to make an orthonormal basis of size $\hat{b} - r$. Denote the new basis as $\hat{Q}_2$, then the following holds,

$$\mathcal{H}V_k = \left[V_k, [Q_1, \hat{Q}_2] \right] \left[\begin{matrix} T_k \\ \begin{bmatrix} [R_{11}, R_{12}]P^{-1}B^T \\ \mathbf{0} \end{bmatrix} \end{matrix} \right]. \tag{7}$$

Therefore the block Krylov–Schur iteration can continue based on (7), with the current block size $\hat{b}$. Note that updating T_k by adding a zero matrix corresponding to the newly added basis Q_2 makes the update economical, any other choice would violate the original Krylov decomposition $\mathcal{H}V_k = V_k T_k + FB^T$ and incur the need to compute residual vectors corresponding to the newly constructed basis.

In our code we assign an integer variable *blkcontrol* to control the change of the block size, *blkcontrol*$=0$ means fixed block size. In the code `block_lanczos()` we do not increase the block size, so if *blkcontrol*$\neq 0$, then the block size decreases whenever F becomes rank deficient. While in `krylov_schur()`, the block size can increase when necessary if the input from the main program is *blkcontrol*$=1$, otherwise the block size output from `block_lanczos()` is kept fixed.

Choosing a best $\hat{b} > b$ when the previous block size b is determined to be too small is a nontrivial task, [3, 36] presented some adaptive techniques. We actually take a more straightforward approach here: we use deflation and reorthogonalization, including reorthogonalization to the deflated converged eigenvectors; by this we can find all the required eigenpairs inside a cluster even when the updated $\hat{b}$ is less than the true size of the eigen cluster.

The other coding detail is that the LAPACK routine *dgeqp*3() for the pivoted QR decomposition (same for the function *qr*() in Matlab) does not produce a permutation matrix P as listed in (5), only a vector containing the permutation information is returned. That is, we only get $F(:, \mathbf{e}) = QR$ where the vector $\mathbf{e}$ contains the column pivoting information, but we need a factorization of the original F. Notice that as long as we get the reverse permutation array of $\mathbf{e}$, i.e., a vector denoted as $\mathbf{inve}$ s.t. $\mathbf{e}(\mathbf{inve}(j)) = j$ for $j = 1 : size(F, 2)$, then it is easy to verify that $F = QR(:, \mathbf{inve})$. Hence the P^{-1} expression in (4) and (5–7) is not a problem.

We note that for a truly robust implementation, a rank revealing pivoted QR decomposition as in [4] should be used, because the pivoted QR implemented in LAPACK *dgeqp*3() and Matlab *qr*() is not guaranteed to give an accurate numerical rank s.t. $rank(R)=rank(F)$. One would have to determined $rank(F)$ separately from the pivoted QR decomposition, which may be costly. Currently our Fortran code uses *dgeqp*3() and Matlab code uses *qr*() directly, this is not the best solution. But the approaches we discussed for the rank deficient cases should be straightforward to implement when a rank revealing pivoted QR decomposition routine becomes available in the future release of LAPACK.

2.3 Deflation of converged wanted eigenpairs

As mentioned in the introduction, one main advantage of the Krylov–Schur method is its convenience in deflating converged Ritz vectors. This advantage is preserved in the block Krylov–Schur method. Suppose that the block Krylov decomposition is (8) and the spectral decomposition of T is (9),

$$\mathcal{H}\tilde{V} = \tilde{V}T + FB^T, \tag{8}$$

$$TX = XD. \tag{9}$$

Denote $V = \tilde{V}X$, and let the QR decomposition of F be $F = QR$, then

$$\mathcal{H}V = VD + QRB^T X. \tag{10}$$

We order the spectral decomposition of T so that the Ritz pairs at the beginning parts of V and D approximate the wanted eigenpairs of $\mathcal{H}$ better than the latter parts. If the residual term $RB^T X$ has structure $[\mathbf{0}, \hat{B}^T]$, then (10) can be written as,

$$\mathcal{H}[V_c, V_{nc}] = [V_c, V_{nc}] \begin{bmatrix} D_c & \\ & D_{nc} \end{bmatrix} + Q[\mathbf{0}, \hat{B}^T]. \tag{11}$$

It follows readily that (11) reduces to

$$\begin{aligned} \mathcal{H}V_c &= V_c D_c, \\ \mathcal{H}V_{nc} &= V_{nc} D_{nc} + Q\hat{B}^T. \end{aligned} \tag{12}$$

That is, the converged Ritz vectors V_c and the converged Ritz values D_c are deflated at the beginning parts of V and D respectively. One only needs to work with (12)—the nonconverged (active) part, for the next Krylov–Schur iterations, with the exception that new basis vectors need to be orthogonalized against V_c. Stewart pointed out [33, p. 347] that the deflation in the Krylov–Schur method is closely related to the implicit deflation in [22, pp. 180–183].

It is easy to see that in this block method we can do deflation vector by vector based on the residual norm of each column in $RB^T X$. There is no need to insist that deflation in a block method should be done block by block. The ease of deflation is one of the powerful features of the Krylov–Schur method.

3 Block Lanczos decomposition

Now we focus on the remaining important step of the block Krylov–Schur method, namely the block Lanczos decomposition. As is expected, the popularity of the Lanczos method [15] in the last several decades has resulted in many publications on block variants of the Lanczos method. However, as noted in the introduction, it is still useful to provide detailed discussions of the rank deficient case. In Section 2.1 we presented an approach based on BLAS-3 rank revealing pivoted QR decomposition for the rank deficient case in the

```
ALGORITHM 3.1. Block Lanczos Decomposition with interface
block_lanczos($V$, $T$, $F$, $k$, $k_f$, $b$, $blkcontrol$)

 I.  If ($k == 0$) then
     1. Compute the thin QR decomposition of F: $F = V(:, 1:b)R$;
     2. Compute $F = \mathcal{H}V(:, 1:b)$;
     3. Compute the first diagonal block of T:
        $T(1:b, 1:b) = V(:, 1:b)^T F$;
     4. Compute new residual vectors that are orthogonal to $V(:, 1:b)$,
        $F = F - V(:, 1:b)T(1:b, 1:b)$;  do necessary reorthogonalization
        s.t. $F \perp V(:, 1:b)$.
 II. Do while ($k + b <= k_f$)
     1. Compute rank revealing pivoted thin QR decomposition of F:
        $F(:, \mathbf{e}) = V(:, k+1 : k+b)R$; set $rankF = rank(F)$;
     2. Compute $\mathbf{inve}$ s.t. $\mathbf{e}(\mathbf{inve}(j)) = j$, for $j = 1:b$;
     3. If ($rankF == b$) then set the subdiagonal block of T:
            $T(k+1 : k+b, k-b+1 : k) = R(:, \mathbf{inve})$;
         Else if ( $0 < rankF$ and $blkcontrol \neq 0$ ) then
            Reduce the block size to $b = rankF$; Store only the first
            $rankF$ rows of R in the subdiagonal block of T:
            $T(k+1 : k+rankF, k-b+1 : k) = R(1 : rankF, \mathbf{inve})$;
        Else
        (a) Orthonormalize $V(:, k+rankF+1 : k+b)$ to $V(:, 1 : k+rankF)$;
        (b) Store the first $rankF$ rows of R in the subdiagonal block of
            T: $T(k+1 : k+rankF, k-b+1 : k) = R(1 : rankF, \mathbf{inve})$;
        (c) Set $T(k+rankF+1 : k+b, k-b+1 : k) = zeros(b-rankF, b)$;
     4. Update the current basis size: $k = k+b$;
     5. Compute $F = \mathcal{H}V(:, k-b+1 : k)$;
     6. Compute $\mathbf{h} = V(:, 1 : k)^T F$;
     7. Get the diagonal block of T:
        $T(k-b+1 : k, k-b+1 : k) = \mathbf{h}(k-b+1 : k, :)$;
     8. Compute new residual vectors that are orthogonal to $V(:, 1 : k)$,
        $F = F - V(:, 1 : k)\mathbf{h}$;  do necessary reorthogonalization s.t.
        $F \perp V(:, 1 : k)$.
     End Do.
```

Fig. 3 Pseudocode for a Block Lanczos decomposition

Krylov–Schur decomposition. This approach is also used to handle the possible rank deficiency in the block Lanczos decomposition.

Pseudocode (Algorithm 3.1) describes how to expand a size-k general Lanczos decomposition of a symmetric $\mathcal{H}$ to a size-k_f general Lanczos decomposition (Fig. 3). That is, starting from

$$\begin{aligned} \mathcal{H}V(:,1:k) &= V(:,1:k)T(1:k,1:k) + FE_k^T, \\ V(:,1:k)^T V(:,1:k) &= I_k, \quad F^T V(:,1:k) = \mathbf{0}, \end{aligned} \tag{13}$$

the code will expand V and T into the following size-k_f decomposition,

$$\begin{aligned} \mathcal{H}V(:,1:k_f) &= V(:,1:k_f)T(1:k_f,1:k_f) + \hat{F}E_{k_f}^T, \\ V(:,1:k_f)^T V(:,1:k_f) &= I_{k_f}, \quad \hat{F}^T V(:,1:k_f) = \mathbf{0}. \end{aligned} \tag{14}$$

The decompositions (13) and (14) are called general Lanczos decomposition because the Rayleigh-quotient matrices T in them are not block tri-diagonal, instead they are of the forms pictured in Fig. 1. By the extra augmentation step in the `krylov_schur()` code, the residual matrix B in a Krylov decomposition is augmented into $T(1{:}k,1{:}k)$ so that the last term in (13) closely resembles a block Lanczos decomposition. One can then call any block Lanczos code to augment (13) into (14). With this design, the coding complexity of Krylov–Schur method is isolated in the `krylov_schur()` part without affecting the block Lanczos part.

In Algorithm 3.1, b denotes the block size. The integer *blkcontrol* is used to control the change of block size: if *blkcontrol*$=0$ then the block size remains constant through out the Lanczos decomposition, otherwise the block size is reduced when F is rank deficient.

We note that the reorthogonalization approach used is full reorthogonalization.

4 Block size constraint of block methods based on Krylov-type decomposition

In this section we briefly discuss the special situation when a large number of good initial vectors are available. This situation is common in the self-consistent calculations based on the density functional theory [13].

An accepted rule on block methods based on Krylov (e.g., Lanczos or Arnoldi) decompositions is that a large block size does not yield good efficiency. Besides hardware concerns (cache size, etc), and the less significant gain for a block method on sparse matrix–vector products, a large block size can potentially reduce the efficiency of a restarted Krylov subspace method. This is because for a restarted method with a fixed maximum subspace dimension, if the maximum degree of the Krylov polynomial for a single vector method is k_{deg}, then the maximum polynomial degree for a method with block-size b is $floor(k_{deg}/b)$, which can be too low to be effective if b is large.

It is tempting to include all the available good initial vectors in the basis V_0 and then augment V_0 using a smaller block size. The problem with this approach is that in the Krylov-type decomposition setting, at the beginning the relation

$$AV_0 = V_0 T + N, \quad \text{where} \;\; T = V_0^T A V_0,$$

needs to be satisfied. The familiar relation $N = FE^T$ for a low rank F orthogonal to V_0 generally does not hold. That is, N needs to have as many columns as V_0 does. Using a block size smaller than the column size of this N will violate the Krylov decomposition and will not result in the desirable $T = V^T AV$ when V_0 is augmented into V. The Davidson-type method that gives up the Krylov subspace structure may be better suited for the task of including a large number of good initial vectors and then using a suitable small block size to augment the basis, see e.g. [29, 30, 37].

5 Numerical results

We developed both Matlab and Fortran codes of the block Krylov–Schur (KS) method. Currently our codes only solve the standard symmetric eigenproblems. For the numerical results in this paper we always compute the *nneed* number of eigenvalues with the smallest algebraic values together with their eigenvectors.

We first test the Matlab code. All the Matlab comparisons are performed on a Dell PC with dual Intel Xeon 2.66G Hz CPU and 1 GB RAM running Debian Linux with Linux kernel version 2.4.25. The Matlab version is 6.5 (R13).

Our Matlab block Krylov–Schur code is compared with two available Matlab codes on block eigensolvers: *irbleigs* of IRBL [1, 2] and *lobpcg* of LOBPCG [11]. As presented in [1, 2], *irbleigs* showed a very good numerical behavior for relatively large symmetric eigenproblems. However, our experience with the code shows that the good numerical behavior is somewhat restricted to the case where a small number of eigenpairs is required. When the required number of eigenpairs increases, the efficiency of *irbleigs* deteriorates. Table 1 shows a comparison of KS, IRBL and LOBPCG (without preconditioning) on a 2-D Laplacian (all Laplacians used in this paper are finite difference discretizations on the unit domain with zero Dirichlet boundary condition).

The block size of KS and IRBL is chosen to be 4. The starting vectors for KS and IRBL are the same random matrix with four columns. IRBL automatically adjusts its block size. To fit the interface of *lobpcg*, this random matrix is augmented into *nneed* columns to be the initial vectors of LOBPCG; that is, the block size of LOBPCG is *nneed*. The maximum restart number is set to 2,500. Table 1 shows that *irbleigs* did not converge within 2,500 restarts for *nneed* $=300$. In several test runs, *irbleigs* often has convergence problem for *nneed* >90 if the maximum restart number is less than 2,000. In contrast, KS converged easily and encountered no problem with increasing *nneed*. Table 1 shows that KS computed 300 eigenpairs in much less cputime than it took

Table 1 Accuracy and cpu (seconds) comparison on a 2-D Laplacian

nneed	15	45	90	105	135	150	300
irbl err.	1.89e-6	1.77e-7	1.28e-6	2.17e-6	1.54e-6	1.23e-6	–
lobpcg err.	3.82e-8	4.58e-9	5.28e-10	3.07e-9	6.04e-10	2.29e-9	7.68e-10
ks err.	1.30e-8	1.33e-8	6.36e-9	5.33e-9	9.15e-10	1.23e-9	7.83e-10
irbl cpu	21.30	213.56	672.86	935.78	1,488.49	1,931.58	–
lobpcg cpu	53.18	108.47	309.23	322.84	483.32	508.28	2,036.24
ks cpu	14.06	20.62	37.97	49.40	78.92	84.82	214.34

$ndim = (70 \times 70) = 4{,}900$. *nneed* is the number of required eigenpairs. We compute the smallest algebraic values. The accuracy (err.) is the largest actual residual norm of converged eigenpairs. The difference of each eigenvalue computed by the three methods (not reported here) is constantly smaller than 10^{-14}.

IRBL and LOBPCG to compute 90 eigenpairs. We also tried the more difficult 1-D Laplacian with dimension 2,000, in this case IRBL did not converge for $nneed > 15$ within 2,500 restarts. LOBPCG and KS both converged to similar accuracy, with KS being at least five times faster than LOBPCG for increasing *nneed*. The efficiency deterioration of *irbleigs* for a large *nneed* may be related to the fact that *irbleigs* automatically adjusts its block size to a large value according to *nneed*, and this large block size is fixed during the iteration, which makes the block Lanczos update less efficient; moreover, it may be difficult to compute a relatively large number of approximate Leja points. We also notice that IRBL computes eigenvalues with same high accuracy as LOBPCG and KS, but the eigenvectors computed by *irbleigs* do not have as high accuracy as LOBPCG and KS. LOBPCG turns out to be more efficient than IRBL for larger *nneed*. One possible reason that LOBPCG becomes less efficient than KS for larger *nneed* is that the local optimal is computed from a subspace of dimension $3 * nneed$, and this "local" subspace is larger than the global subspace (dimension $\approx 2 * nneed$) used in KS.

In the following we compare our Matlab code with the Matlab v6.5 *eigs*. This *eigs* function is essentially the Fortran ARPACK accessed via the Matlab mex interface, hence it is very efficient and well tuned. While it is not realistic to expect codes written directly in Matlab to compare favorably with *eigs* in cputime, especially for relatively large eigenproblems with a not too small *nneed*, our goal is to see if a block method can have some advantages. We use two silicon quantum dot models from materials science, $Si_{10}H_{16}$ and SiO, where $Si_{10}H_{16}$ is the hydrogenated silicon, with ten silicon atoms passivated by 16 hydrogen atoms, and SiO is an monoxidized silicon. The matrices are obtained from ab initio DFT calculations after the self-consistency is reached [5, 14]. Figure 4 shows the sparsity structures of the corresponding Hamiltonian matrices. The largest multiplicity for the computed eigenvalues of $Si_{10}H_{16}$ and SiO is 3 and 2 respectively.

For the numerical test, the *blksize* is set to 3. We first compute *blksize* converged eigenpairs (V, D) by *eigs*, then apply perturbations to V by using

$$V_0 = V(:, 1 : blksize) + 10^{-i} * rand(ndim, blksize), \quad \text{for} \quad i = 2 : 10 \qquad (15)$$

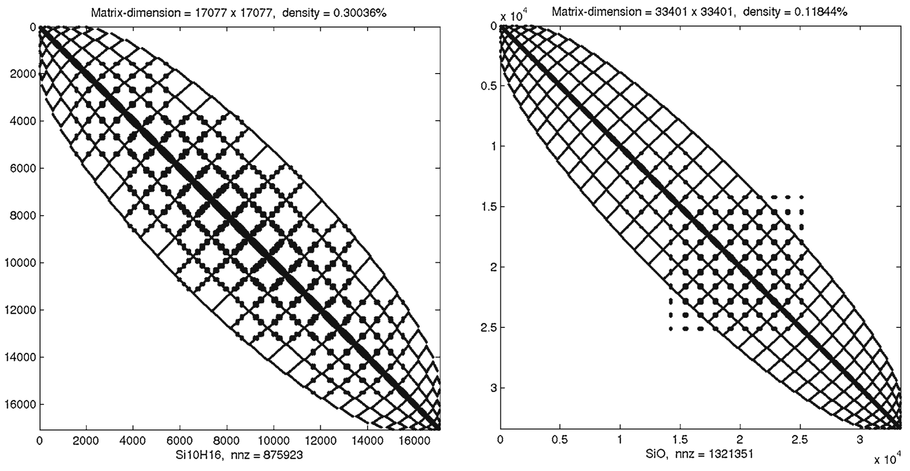


Fig. 4 Sparsity structures of the Hamiltonian matrices of the silicon quantum dots $Si_{10}H_{16}$ and SiO, with dimension 17,077 and 33,401 respectively

as the initial three vectors for the block Krylov–Schur method. The column sums of V_0 is used as the initial vector for *eigs*. The convergence tolerance is set to 10^{-11}. We compute $nneed = 15, 25, 35$ eigenpairs. The contracted subspace dimension k_s is set to *nneed*; and the augmented subspace dimension k_f is set to $2{*}nneed$, which is extended slightly after deflation by each method.

Figure 5 shows the cputime results for $Si_{10}H_{16}$ and SiO, respectively. It may seem surprising that the improved initial vectors do not necessarily lead to faster overall convergence. However, this is not surprising because *nneed* is at least five times larger than the number of good initial vectors. If we set $blksize = nneed$, then as predicted, the improved initial vectors lead to less iteration steps to convergence for the block method, but $blksize = nneed$ actually takes longer cputime to converge than with $blksize = 3$ except when the perturbation in (15) becomes smaller than 10^{-9}. It may also seem surprising that for both models, $nneed = 15$ usually takes longer to converge than $nneed = 25$. This is not surprising if we note that the augmented subspace dimension is set to $2{*}nneed$, which means that the degree of the Krylov polynomial for $nneed = 15$ is smaller than that for $nneed = 25$. As seen from both Fig. 5, the cputime for the Matlab block Krylov–Schur code is comparable with the Matlab *eigs*. For these two tests, Krylov–Schur uses more cputime with $blksize = 1$ than with $blksize = 3$, this confirms the advantage of the block method. We also note that for $nneed > 50$ the *eigs* which is essentially based on a Fortran package wins more in cputime.

Finally we present comparisons between our Fortran block Krylov–Schur code and the Fortran ARPACK. The first example is a 2-D Laplacian with $ndim$ = 62,500, 78,400 and 90,000. The block size is set to 4.

As can be seen from Table 2, even though the restart numbers and matrix–vector multiplications of the block Krylov–Schur both exceed those of

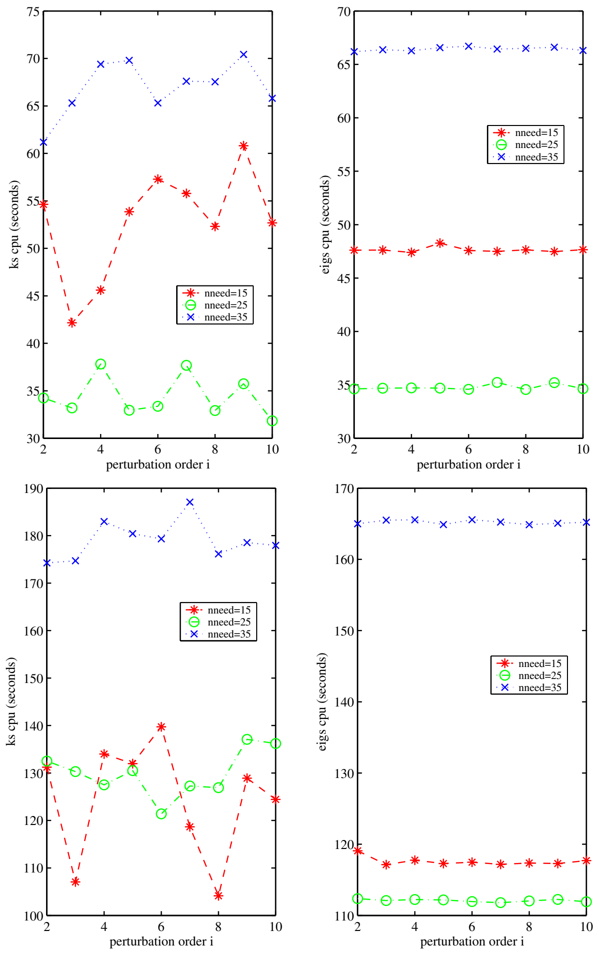


Fig. 5 Cputime comparison for block Krylov–Schur (with three improved initial vectors) and *eigs* (with one improved initial vector) for $Si_{10}H_{16}$ (*top*) and SiO (*bottom*)

ARPACK, the cputime of the block method is smaller. The gain in cputime can be attributed to the fact that the eigenvalues of the 2-D Laplacian of the tested dimensions are not well separated, and that the block size 4 fits the multiplicity of the eigenvalues, hence the block method has an advantage. The computation is done on the IBM power4 supercomputer running AIX 5.2 of the Minnesota Supercomputing Institute. All codes are compiled by the IBM Fortran compiler *xlf_r* using optimization flags: `-O4 -qstrict -qmaxmem=-1 -qtune=pwr4 -qarch=pwr4`.

We also test our codes on two larger Silicon quantum dot models: $Si_{34}H_{36}$ and an oxidized silicon SiO_2. The sparsity structures and dimensions of the Hamiltonian matrices are shown in Fig. 6. The eigenvalues for these models are not very clustered, actually the computed *nneed* eigenvalues of SiO_2 are all simple. Thus ARPACK is expected to be very efficient for these models.

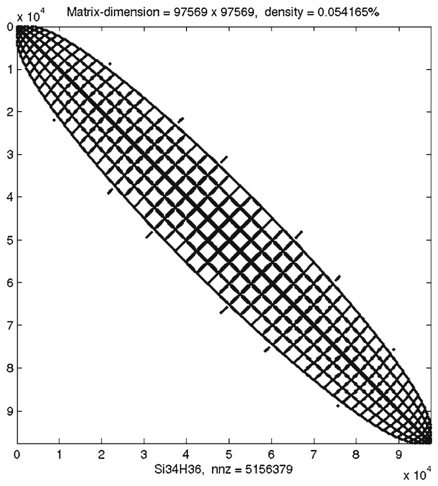
Table 2 Comparison of Fortran block Krylov–Schur code with ARPACK on a 2-D Laplacian with different *ndim*

	# mat-vec mult	# restart	cpu (seconds)
ndim = 62,500			
ARPACK	4,935	31	853.81
KS	5,856 (1,464)	33	691.48
ndim = 78,400			
ARPACK	5,469	34	1,289.69
KS	6,672 (1,668)	38	1,040.28
ndim = 90,000			
ARPACK	5,912	37	1,607.77
KS	7,132 (1,783)	41	1,257.10

nneed = 100. *tol* = 10^{-12}. Maximum subspace dimension is set to 300. All the source codes are compiled with the same optimization flags. The numbers in parenthesis correspond to the number of block matrix–vector multiplications.

On the IBM AIX power4 supercomputer ARPACK is constantly faster than the block Krylov–Schur code. However, the block Krylov–Schur code is faster than ARPACK on a Sun Blade-2000 workstation that has 6 GB RAM, using one of the dual 900 Mhz UltraSparcIII cpus. The compiler used on the Sun workstation is Sun f90 compiler, with optimization flags: `-64 -fast -dalign -native -xO5` (Table 3).

The comparisons between our Fortran block Krylov–Schur package and ARPACK are preliminary, but we expect that a well developed package based on the block version of the Krylov–Schur algorithm will be quite useful in some situations such as when the eigenvalues have high multiplicity or are clustered.



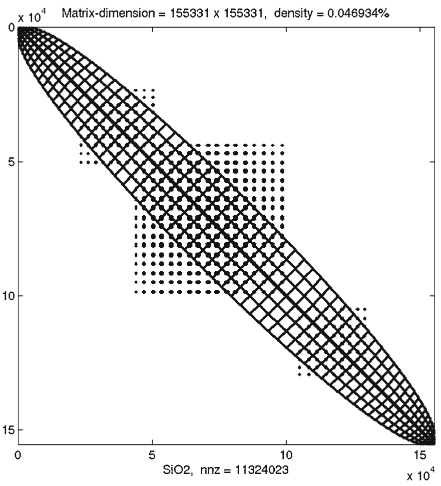


Fig. 6 Sparsity structures of the Hamiltonian matrices of the silicon quantum dots $Si_{34}H_{36}$ and SiO_2, with dimension 97,569 and 155,331 respectively

Table 3 Comparison of Fortran block Krylov–Schur code with ARPACK on $Si_{34}H_{36}$ and SiO_2

	# mat-vec mult	# restart	cputime (IBM)	cputime (SUN)
ndim = 97,569				
ARPACK	1,424	17	235.30 s	1,092.70 s
KS	1,719 (573)	19	271.85 s	960.01 s
ndim = 155,331				
ARPACK	1,652	22	573.65 s	2,347.08 s
KS	2,103 (701)	26	678.01 s	2,194.42 s

$nneed = 50$, $blksize = 3$, $tol = 10^{-12}$. Maximum subspace dimension is set to 150. The numbers in parenthesis correspond to the number of block matrix–vector multiplications. On each platform all the source codes are compiled with the same optimization flags.

6 Conclusions

In this paper we studied an extension of the Krylov–Schur algorithm into a block version. We proposed to use the rank revealing pivoted QR to handle the rank deficient cases that can cause difficulties in block methods. We also introduced ways to adaptively adjust the block size during iterations inside the block Krylov–Schur algorithm. We developed a Matlab code of the block Krylov–Schur method, and the numerical comparisons show that, this code is efficient and robust. We also developed a Fortran package based on the block Krylov–Schur method. The preliminary comparisons with the Fortran package ARPACK are encouraging.

References

1. Baglama, J., Calvetti, D., Reichel, L.: IRBL: an implicitly restarted block-lanczos method for large-scale Hermitian eigenproblems. SIAM J. Sci. Comput. **24**, 1650–1677 (2003)
2. Baglama, J., Calvetti, D., Reichel, L.: Irbleigs: a MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix. ACM Trans. Math. Software **5**, 337–348 (2003)
3. Bai, Z., Day, D., Ye, Q.: ABLE: an adaptive block Lanczos method for non-Hermitian eigenvalue problems. SIAM J. Matrix Anal. Appl. **20**, 1060–1082 (1999)
4. Bischof, C.H., Quintana-Ortí, G.: Computing rank-revealing QR factorizations of dense matrices. ACM Trans. Math. Software **24**, 226–253 (1998)
5. Chelikowsky, J.R., Troullier, N., Saad, Y.: Finite-difference-pseudopotential method: electronic structure calculations without a basis. Phys. Rev. Lett. **72**, 1240–1243 (1994)
6. Dai, H., Lancaster, P.: Preconditioning block Lanczos algorithm for solving symmetric eigenvalue problems. J. Comput. Math. **18**(4), 365–374 (2000)
7. Daniel, J., Gragg, W.B., Kaufman, L., Stewart, G.W.: Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. Math. Comp. **30**, 772–795 (1976)
8. Freund, R.W.: Band Lanczos method. In: Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H. (eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, pp. 80–88. SIAM, Philadelphia (2000)
9. Golub, G.H., Luk, F.T., Overton, M.L.: A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. ACM Trans. Math. Software **7**, 149–169 (1981)
10. Grimes, R.G., Lewis, J.G., Simon, H.D.: A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. SIAM J. Matrix Anal. Appl. **15**, 228–272 (1994)
11. Knyazev, A.V.: Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. **23**(2), 517–541 (2001)

12. Knyazev, A.V., Neymeyr, K.: Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method. ETNA **15**, 38–55 (2003)
13. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. **140**, A1133–A1138 (1965)
14. Kronik, L., Makmal, A., Tiago, M., Alemany, M., Jain, M., Huang, X., Saad, Y., Chelikowsky, J.: PARSEC—the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures. Phys. Status Solidi B **243**, 1063–1079 (2006)
15. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Nat. Bur. Standards **45**, 255–282 (1950)
16. Lehoucq, R., Sorensen, D.C.: Deflation techniques for an implicitly restarted Arnoldi iteration. SIAM J. Matrix Anal. Appl. **17**, 789–821 (1996)
17. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK user's guide: solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM, Philadelphia. Available at http://www.caam.rice.edu/software/ARPACK/ (1998)
18. Miminis, G., Paige, C.: Implicit shifting in the QR and related algorithms. SIAM J. Matrix Anal. Appl. **12**, 385–400 (1991)
19. Parlett, B.N.: The symmetric eigenvalue problem. No. 20 in Classics in Applied Mathematics. SIAM, Philadelphia, PA (1998)
20. Parlett, B.N., Le, J.: Forward instability of tridiagonal QR. SIAM J. Matrix Anal. Appl. **4**, 279–316 (1993)
21. Saad, Y.: On the rates of convergence of the Lanczos and the block Lanczos methods. SIAM J. Numer. Anal. **17**, 687–706 (1980)
22. Saad, Y.: Numerical methods for large eigenvalue problems. John Wiley, New York. Available at http://www.cs.umn.edu/~saad/books.html (1992)
23. Sadkane, M.: Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems. Numerische Mathematik **64**, 195–212 (1993a)
24. Sadkane, M.: A block Arnoldi–Chebyshev method for computing leading eigenpairs of large sparse unsymmetric matrices. Numerische Mathematik **64**, 181–194 (1993b)
25. Sadkane, M., Sidje, R.B.: Implementation of a variable block Davidson method with deflation for solving large sparse eigenproblems. Numer. Algor. **20**(2), 217–240 (1999)
26. Scott, D.S.: Block Lanczos software for symmetric eigenvalue problems. Technical report ORNL/CSD-48, Oak Ridge National Laboratory, Oak Ridge, TN (1979)
27. Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method. SIAM J. Matrix Anal. Appl. **13**, 357–385 (1992)
28. Sorensen, D.C.: Deflation for implicitly restarted Arnoldi methods. Technical report CAAM:TR98-12, Rice University (1998)
29. Stathopoulos, A.: Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: seeking one eigenvalue. SIAM J. Sci. Comput. **29**(2), 481–514 (2007)
30. Stathopoulos, A., McCombs, J.R.: Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: seeking many eigenvalues. SIAM J. Sci. Comput. **29**(5), 2162–2188 (2007)
31. Stathopoulos, A., Saad, Y., Wu, K.: Dynamic thick restarting of the Davidson and the implicitly restarted Arnoldi methods. SIAM J. Sci. Comput. **19**, 227–245 (1998)
32. Stewart, G.W.: A Krylov–Schur algorithm for large eigenproblems. SIAM J. Matrix Anal. Appl. **23**, 601–614 (2001)
33. Stewart, G.W.: Matrix Algorithms II: Eigensystems. SIAM, Philadelphia (2001)
34. Watkins, D.S.: Forward stability and transmission of shifts in the QR algorithm. SIAM J. Matrix Anal. Appl. **16**, 469–487 (1995)
35. Wu, K., Simon, H.: Thick-restart Lanczos method for large symmetric eigenvalue problems. SIAM J. Matrix Anal. Appl. **22**, 602–616 (2000)
36. Ye, Q.: An adaptive block Lanczos algorithm. Numer. Algor. **12**, 97–110 (1996)
37. Zhou, Y.: A block Chebyshev–Davidson method with inner-outer restart for large eigenvalue problems. Technical report, Southern Methodist University (to be submitted)