UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Scalable Library for Eigenvalue Problem Computations SLEPc

SLEPc Technical Report **STR-7**
Available at http://slepc.upv.es

Krylov-Schur Methods in SLEPc

V. Hernández
J. E. Román
A. Tomás
V. Vidal

Last update: June, 2007 (SLEPc 2.3.3)
Previous updates: –

About SLEPc Technical Reports: These reports are part of the documentation of SLEPc, the *Scalable Library for Eigenvalue Problem Computations*. They are intended to complement the Users Guide by providing technical details that normal users typically do not need to know but may be of interest for more advanced users.

Contents

1 Introduction

The Krylov-Schur method was introduced in 2001 by Stewart and can be seen as an improvement on traditional Krylov subspace methods such as Arnoldi and Lanczos. More precisely, the Krylov-Schur method incorporates an effective and robust restarting scheme, resulting in fast convergence in most cases.

Although this report is basically self-contained, the reader is recommended to consult also the SLEPc Technical Reports STR-4, "Arnoldi Methods in SLEPc", and STR-5, "Lanczos Methods in SLEPc", since all three methods share several common aspects both theoretically and in the implementation. Remember that Arnoldi is a general method that can address non-Hermitian problems, whereas Lanczos is restricted to Hermitian ones. In the case of the Krylov-Schur method, SLEPc provides a single solver for both Hermitian and non-Hermitian problems. In the description of the algorithm below, those steps of the algorithm that may be different depending on whether symmetry is taken into account or not will be highlighted.

Section 2 provides a general description of the method. Then section 3 gives some details that are particular to the SLEPc implementation. Also, this report includes some performance results in various platforms (section 4).

2 Description of the Method

The main reference for the Krylov-Schur method is [Stewart, 2001], together with its addendum [Stewart, 2002]. The interested reader is referred to these references for a detailed exposition. A description of the algorithm can also be found in other sources such as [Kressner, 2005].

2.1 Overview of Krylov Projection Methods

Projection methods for eigenvalue problems are intended for computing a partial eigensolution, that is, given a square matrix A of order n, the objective is to compute a small number of eigenpairs, λ_i, x_i, $i = 1, \ldots, k$, with $k \ll n$. The basic principle of projection methods is to find the best approximations to the eigenvectors in a given subspace of small dimension. This is achieved with the so-called Rayleigh-Ritz procedure, described next.

Given an $n \times m$ matrix V, with $k \leq m \ll n$, whose columns v_i constitute an orthonormal basis of a given subspace $\mathcal{V}$, i.e. $V^T V = I_m$ and $\text{span}\{v_1, v_2, \ldots, v_m\} = \mathcal{V}$, the Rayleigh-Ritz procedure consists in computing $H = V^T AV$, which is a square matrix of order m, and then solving the eigenvalue problem associated with it, $Hy_i = \theta_i y_i$. The approximate eigenpairs $\tilde{\lambda}_i, \tilde{x}_i$ of the original eigenvalue problem are then $\tilde{\lambda}_i = \theta_i$ and $\tilde{x}_i = Vy_i$, which are called Ritz values and Ritz vectors, respectively. Note that Ritz vectors belong to subspace $\mathcal{V}$. It can be shown that the Ritz values and vectors are the best possible approximations in that subspace. Matrix H is the projection of A onto $\mathcal{V}$, hence the name of this class of methods.

The above process works better when the subspace $\mathcal{V}$ approximates a certain invariant subspace of A. This is the reason why the Rayleigh-Ritz projection is often combined with the simultaneous iteration algorithm, in which case $\mathcal{V} = \text{span}(A^s X_1)$ for a given set of m initial vectors X_1 and increasing s. However, the overall process is much more effective if $\mathcal{V}$ is equal to the Krylov subspace associated with matrix A and a given initial vector x_1,

$$\mathcal{K}_m(A, x_1) = \text{span}\{x_1, Ax_1, A^2x_1, \ldots, A^{m-1}x_1\} \,. \tag{1}$$

The method of Arnoldi is a Krylov-based projection algorithm that computes an orthogonal basis of the Krylov subspace and at the same time computes the projected matrix H, all this in an efficient and numerically stable way.

Algorithm 1 (Basic Arnoldi)

Input: Matrix A, number of steps m, and initial vector v_1 of norm 1
Output: (V_m, H_m, f, β) so that $AV_m = V_m H_m + fe_m^*$, $\beta = \|f\|_2$

```
For j = 1, 2, ..., m − 1
    w = Av_j
    Orthogonalize w with respect to V_j (obtaining h_{i,j} for i = 1, ..., j)
    h_{j+1,j} = ‖w‖_2    (if h_{j+1,j} = 0, stop)
    v_{j+1} = w/h_{j+1,j}
end
f = Av_m
Orthogonalize f with respect to V_m (obtaining h_{i,m} for i = 1, ..., m)
β = ‖f‖_2
```

In Algorithm 1, a subindex is used in H and V to make their dimension explicit. Note that in each iteration of the algorithm, a new column of both H and V is computed. In the case of Arnoldi, H has an unreduced upper Hessenberg form, that is, upper triangular with an

additional non-zero subdiagonal. The orthogonalization operations in Algorithm 1 are typically carried out by means of a Gram-Schmidt procedure, that removes all the components in the directions of the Arnoldi vectors, $v_1, \ldots, v_j$. The quantities computed by the algorithm satisfy the following relation

$$AV_m = V_m H_m + f e_m^*, \tag{2}$$

which is usually called m-step Arnoldi factorization or Arnoldi decomposition of order m. The last computed vector f, called the residual, gives an indication of how close is $\mathcal{V}$ to an invariant subspace. In particular, its norm β is used to assess the accuracy of the computed Ritz pairs, since

$$\|A\tilde{x}_i - \tilde{\lambda}_i \tilde{x}_i\|_2 = \|AV_m y_i - \theta_i V_m y_i\|_2 = \|(AV_m - V_m H_m) y_i\|_2 = \beta |e_m^* y_i|. \tag{3}$$

With the Arnoldi process, the Ritz pairs will converge very fast provided that the initial vector x_1 is rich in the direction of the wanted eigenvectors. However, in practice this is usually not the case and consequently many iterations are likely to be required. This is a serious problem because increasing the number of iterations (m) implies a growth in storage requirements and, more importantly, a growth of computational cost per iteration. A workaround for this is to do a *restart* of the algorithm, that is, stop after m iterations and rerun the algorithm with a new x_1 computed from the recently obtained spectral approximations. One possible approach is to explicitly compute x_1 as a linear combination of a subset of the computed Ritz vectors. This is called explicit restart. Other variants of explicit restart additionally apply a polynomial in A of degree d, $\psi(A)$, to the initial vector with the intent to filter out components in unwanted directions.

The main difficulty in explicit restart is how to choose the parameters for building the new starting vector. In the last years, an important research effort has been devoted to improving the effectiveness of restarting in the context of Krylov eigensolvers, by eliminating the need to explicitly compute a new vector x_1.

Implicit restart is an alternative in which the Arnoldi process is combined with the implicitly shifted QR algorithm. An m-step Arnoldi factorization is compacted into an $(m-d)$-step Arnoldi factorization, which is then extended again to an m-step one. The key point is that the small factorization retains the relevant eigeninformation of the large factorization, and this is accomplished by applying several steps of the QR iteration. This process is more efficient than explicit restart, and can be interpreted as an implicit application of a polynomial in A of degree d to the initial vector, [Sorensen, 1992]. The implicit restart technique has been implemented in the ARPACK software, [Lehoucq *et al.*, 1998].

2.2 The Krylov-Schur Method

Implementing the implicit restart technique in a numerically stable way is difficult. A simpler way of achieving the same effect is by the Krylov-Schur method proposed by Stewart [2001].

The Krylov-Schur method is defined by generalizing the Arnoldi decomposition of order m,

$$AV_m = V_m H_m + \beta v_{m+1} e_m^* \,, \tag{4}$$

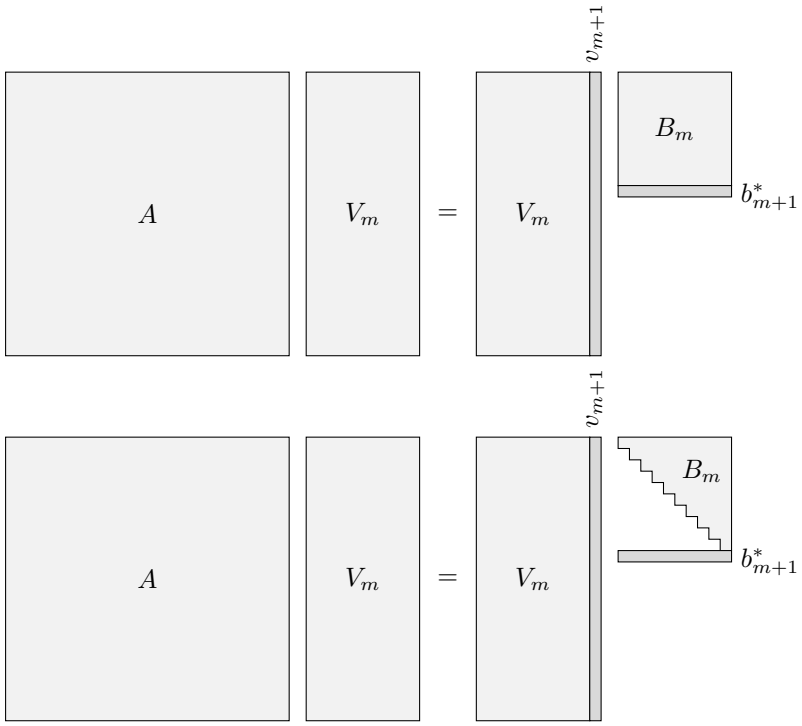

Figure 1: Scheme of a general Krylov decomposition (above) and a Krylov-Schur decomposition (below).

computed by Algorithm 1, where $v_{m+1} = f/\beta$, to a so-called Krylov decomposition of order m,

$$AV_m = V_m B_m + v_{m+1} b^*_{m+1} \,, \tag{5}$$

in which matrix B_m is not restricted to be upper Hessenberg and b_{m+1} is an arbitrary vector. Assuming that all the v_i vectors are mutually orthonormal, pre-multiplying Eq. 5 by V^*_m shows that B_m is the Rayleigh quotient $V^*_m A V_m$ so the Rayleigh-Ritz procedure is still valid.

The last equation is equivalent to

$$AV_m = \begin{bmatrix} V_m & v_{m+1} \end{bmatrix} \begin{bmatrix} B_m \\ b^*_{m+1} \end{bmatrix} . \tag{6}$$

A special case of the above relation is the Krylov-Schur decomposition, in which matrix B_m is in real Schur form, that is, quasi-triangular form displaying eigenvalues in the 1×1 or 2×2 diagonal blocks. Figure 1 shows a graphical scheme of these factorizations. Note that for simplicity the picture shows only 1×1 diagonal blocks.

The Arnoldi decomposition is a particular case of the Krylov decomposition. It can be shown [Stewart, 2001, Th. 2.2] that any Krylov decomposition is *equivalent* to an Arnoldi decomposition, that is, it has the same Ritz approximations. Also, it is possible to transform a Krylov decomposition into an equivalent Krylov-Schur decomposition by means of orthogonal similarity transformations. The basic idea of the Krylov-Schur method is to iteratively expand (with the Arnoldi process) and contract a Krylov-Schur decomposition. A schematic description of the method is given in Algorithm 2.

Algorithm 2 (Krylov-Schur Method)

Input: Matrix A, initial vector x_1, and number of steps m
Output: $k \leq p$ Ritz pairs

1. Build an initial Krylov decomposition of order m
2. Apply orthogonal transformations to get a Krylov-Schur decomposition
3. Reorder the diagonal blocks of the Krylov-Schur decomposition
4. Truncate to a Krylov-Schur decomposition of order p
5. Extend to a Krylov decomposition of order m
6. If not satisfied, go to step 2

Step 1 can be accomplished for instance with an Arnoldi decomposition with Algorithm 1. For step 2, it is necessary to apply the QR algorithm in order to compute an orthogonal matrix Q_1 such that $T_m = Q_1^* B_m Q_1$ has real Schur form, and then

$$AV_mQ_1 = \begin{bmatrix} V_mQ_1 & v_{m+1} \end{bmatrix} \begin{bmatrix} T_m \\ b_{m+1}^* Q_1 \end{bmatrix} . \tag{7}$$

At this point, the Ritz values are readily available from the diagonal blocks of T_m. These Ritz values are divided in two subsets: Ω_w containing $p < m$ "wanted" Ritz values and Ω_u containing $m - p$ "unwanted" Ritz values. The objective of step 3 is to move the subset of wanted Ritz values to the leading principal submatrix of T_m. This can also be accomplished by means of an orthogonal transformation Q_2, resulting in the following reordered Krylov-Schur decomposition

$$A\tilde{V}_m = \begin{bmatrix} \tilde{V}_m & v_{m+1} \end{bmatrix} \left[\begin{array}{cc} T_w & \star \\ 0 & T_u \\ \hline b_w^* & \star \end{array}\right] , \tag{8}$$

where $\tilde{V}_m = V_mQ_1Q_2$, $\lambda(T_w) = \Omega_w$, $\lambda(T_u) = \Omega_u$, and b_w^* is the length-p leading subvector of $b_{m+1}^* Q_1 Q_2$. The truncation in step 4 of the algorithm is achieved simply by writing

$$A\tilde{V}_p = \begin{bmatrix} \tilde{V}_p & \tilde{v}_{p+1} \end{bmatrix} \begin{bmatrix} T_w \\ b_w^* \end{bmatrix} , \tag{9}$$

where $\tilde{V}_p$ is equal to the first p columns of $\tilde{V}_m$, but $\tilde{v}_{p+1} = v_{m+1}$. This step is illustrated in Figure 2, in which the parts to be removed from the factorization are shadowed.

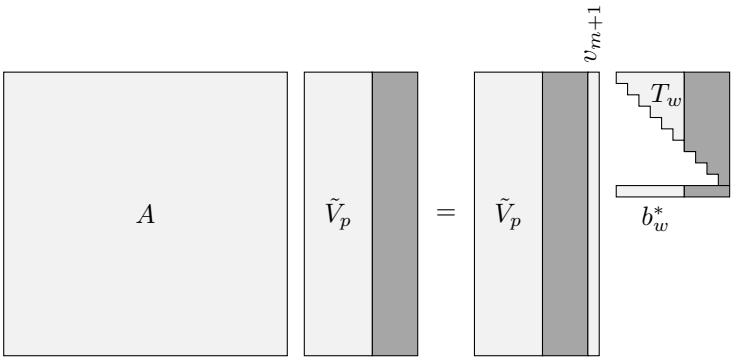

Figure 2: Scheme of the truncation step in the reordered Krylov-Schur decomposition.

Figure 3: Scheme of matrix B_m after step 5 (before reduction to the Krylov-Schur form), in the non-symmetric case (left) and the symmetric case (right).

Finally, step 5 is carried out by means of a variation of Algorithm 1 in which the vectors are computed starting from v_{p+2} but vectors v_1 to v_{p+1} are also taken into account in the orthogonalization step. For example, Figure 3 (left) shows the shape of matrix B_m after expanding the truncated matrix of order p to order m. In the picture, 4 new columns have been added (the shadowed part), corresponding to four steps of the Arnoldi method with initial vector v_{m+1}.

2.3 Symmetric Krylov-Schur

In the case that matrix A is symmetric (or Hermitian), some additional considerations can be done. First of all, matrix H_m of the Arnoldi decomposition (2) is also symmetric and thus tridiagonal. Also, zeros in the upper part of H_m indicate that orthogonality need not be enforced explicitly against all the v_j vectors, at least in exact arithmetic (this is not the case in finite precision arithmetic). These special characteristics are exploited by the Lanczos method, a very simple and elegant algorithm that may be viewed as a specialized variant of Arnoldi. A more comprehensive description of the Lanczos method and its implementation in SLEPc can be found in the SLEPc Technical Report STR-5, "Lanczos Methods in SLEPc".

The Krylov-Schur method can be adapted similarly to the symmetric case. When the symmetries in the Krylov-Schur decomposition are considered, one soon realizes that the Krylov-

Schur method is equivalent in the symmetric case to another method that was proposed before: the *thick-restart* Lanczos method, [Wu and Simon, 2000].

The most significant difference between the symmetric and non-symmetric cases is the shape of matrix B_m. Figure 3 illustrates this difference graphically. The right part of the figure shows the characteristic arrow-head shape that is obtained in the symmetric case, which then grows from below by a number of Lanczos steps.

From the implementation point of view, the structure of this matrix B_m poses more difficulties than in the case of a simple explicitly restarted variant of Lanczos, where the projected matrix consisted of a diagonal part and a tridiagonal part. In thick-restart Lanczos, obtaining the Krylov-Schur form of B_m is equivalent to diagonalizing it. For this, taking into account its special structure may suppose a marginal reduction in the operation count for standard solvers whose first stage is tridiagonalization. Therefore, it is appropriate to use a general dense symmetric method (note that this is not a big issue due to the small dimension of this matrix). An alternative is to devise a specific algorithm for this particular structure, although it is not obvious how to do this.

The other thing to be considered in the symmetric case is whether orthogonalization of the Lanczos vectors should be carried out explicitly against all previous vectors, or, on the contrary, it is worth using a technique for the cure of loss of orthogonality (such as partial orthogonalization). In the context of thick-restart Lanczos, the conclusion is that full orthogonalization is the best option, since it provides the maximum robustness (which may favor the convergence of the method) at a moderate additional cost (in this context, orthogonalization with respect to the first p vectors is required anyway).

2.4 Other Variants

This subsection describes very briefly some variations of the algorithm that may be of interest in some situations.

Generalized Problems. When addressing a generalized eigenvalue problem, $Ax = \lambda Bx$, usually a spectral transformation is used. In the symmetric case, symmetry is lost but can be recovered by employing the so-called B-Lanczos process, which amounts to replacing the standard Hermitian inner product, $\langle x, y\rangle = y^*x$, by the B-inner product, $\langle x, y\rangle_B = y^*Bx$. Similarly, the B-Arnoldi process may provide benefits from the numerical point of view. For a more detailed description of this issue, see SLEPc reports STR-4 and STR-5.

These modifications are also relevant to the Krylov-Schur method, since it is ultimately based on the Arnoldi and Lanczos recurrences.

Block Variant. A block Krylov method tries to extract approximate spectral information from a block Krylov subspace

$$\mathcal{K}_m(A, V_1) = \text{span}\{V_1, AV_1, A^2V_1, \ldots, A^{m-1}V_1\} \quad , \tag{10}$$

where V_1 has b columns. Both Arnoldi and Lanczos can be thought in terms of block Krylov subspaces. Again, the reader is referred to SLEPc reports STR-4 and STR-5 for further discussion. As before, this is also applicable to the Krylov-Schur method, inheriting both advantages and complications. For a more detailed discussion of the block Krylov-Schur method, see [Zhou and Saad, 2004].

2.5 Available Implementations

An implementation of the block Krylov-Schur method is available in the ANASAZI eigensolver package, which is part of TRILINOS, a parallel object-oriented software framework for large-scale multi-physics scientific applications. For additional information, see SLEPc Technical Report STR-6, "A Survey of Software for Sparse Eigenvalue Problems".

3 The SLEPc Implementation

SLEPc provides an implementation of the Krylov-Schur method. The corresponding solver is `EPSKRYLOVSCHUR` (or `-eps_type krylovschur` from the command-line). Algorithm 3 represents a detailed version of Algorithm 2, that intends to be closer to the actual SLEPc implementation.

Algorithm 3 (Krylov-Schur Method)

Input: Matrix A, initial vector v_1, and dimension of the subspace m
Output: A partial Schur decomposition $AV_{1:k} = V_{1:k}H_{1:k,1:k}$

Normalize v_1
Initialize $V_m = [v_1], \quad k = 0, \quad p = 0$
Restart loop
  Perform $m - p$ steps of Arnoldi with deflation
  Reduce H_m to (quasi-)triangular form, $H_m \leftarrow U_1^* H_m U_1$
  Sort the 1×1 or 2×2 diagonal blocks: $H_m \leftarrow U_2^* H_m U_2$
  $U = U_1 U_2$
  Compute eigenpairs of H_m, $H_m y_i = y_i \theta_i$
  Compute residual norm estimates, $\tau_i = \beta |e_m^* y_i|$
  $V_m \leftarrow V_m U$
  Exit if enough converged eigenpairs, otherwise lock newly converged vectors
  Choose p and set $\tilde{v}_{p+1} = v_{m+1}$
  Compute b_w and insert it in the appropriate positions of H_p
end

Selection of p. Currently, the SLEPc implementation sets the value of p to somewhere in-between m (the maximum subspace dimension) and k (the number of currently converged eigenpairs). Its value grows progressively as eigenvalues converge.

3.1 Known Issues and Applicability

The Krylov-Schur method is usually much faster and effective than Arnoldi. Also, both methods behave similarly in terms of robustness. For this reason, it is currently the default method in SLEPc.

Supported problem types	All
Allowed portion of the spectrum	All
Support for complex numbers	Yes

4 Performance Results

In this section we provide the results of some tests to illustrate parallel performance of the Krylov-Schur solver in a typical situation.

The SLEPc Krylov-Schur implementation has been compared with ARPACK in two different computing platforms. The first machine used for the tests is a cluster of 55 dual processor nodes with Pentium Xeon processors at 2.8 GHz interconnected with an SCI network in a 2-D torus configuration. In order to extend the analysis to more processors, the tests were repeated in MareNostrum, an IBM JS21 cluster of 2,560 tetraprocessor nodes with PowerPC 970MP processors at 2.3 GHz interconnected with a Myrinet network. Only one processor per node was used in the tests reported in this section.

The tests consist in measuring the parallel speed-up when computing the 10 largest eigenvalues of two matrices from real-world applications, AF23560 and PRE2. The AF23560 matrix is real non-symmetric of order 23,560 and it is the largest one from the NEP collection [Bai *et al.*, 1997]. The PRE2 matrix is real non-symmetric of order 659,033 from the University of Florida Sparse Matrix Collection [Davis, 1992]. The speed-up is calculated as the ratio of elapsed time with p processors to the elapsed time with one processor corresponding to the fastest algorithm. This latter time corresponds in both cases to the SLEPc Krylov-Schur implementation. In these tests both eigensolvers were configured with tolerance equal to 10^{-7} and a maximum of 30 basis vectors (`ncv`).

Figure 4 shows the speed-up obtained with the two matrices in the Xeon cluster. Due to the small size of the AF23560 matrix, the speed-up begins to flatten at around 30 processors. This happens because the time spent doing computational work by each processor decreases and the time spent doing communication between processors begins to dominate. With the much larger PRE2 matrix this effect does not occur at all. In this cluster the performance of SLEPc is slightly better than ARPACK.

The results with the same matrices in the PowerPC cluster (figure 5) show the same speed-up stagnation when the number the processors is sufficiently large. However, this effect appears only at around 250 processors with the largest matrix, which is a very good parallel performance result. Also, the SLEPc performance is clearly better than ARPACK in this case.

In order to measure parallel performance without the problem-size effect and to be able to analyze scalability to a larger number of processors, a synthetic test case has been used. This test consists in measuring the scaled speed-up with the same solver parameters as before

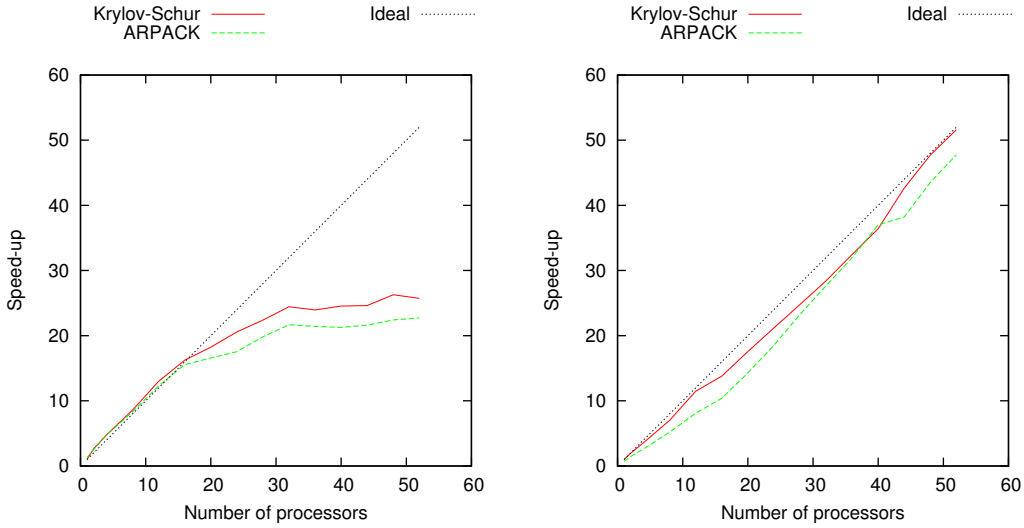

Figure 4: Speed-up in Xeon cluster for AF23560 (left) and PRE2 (right) matrices.

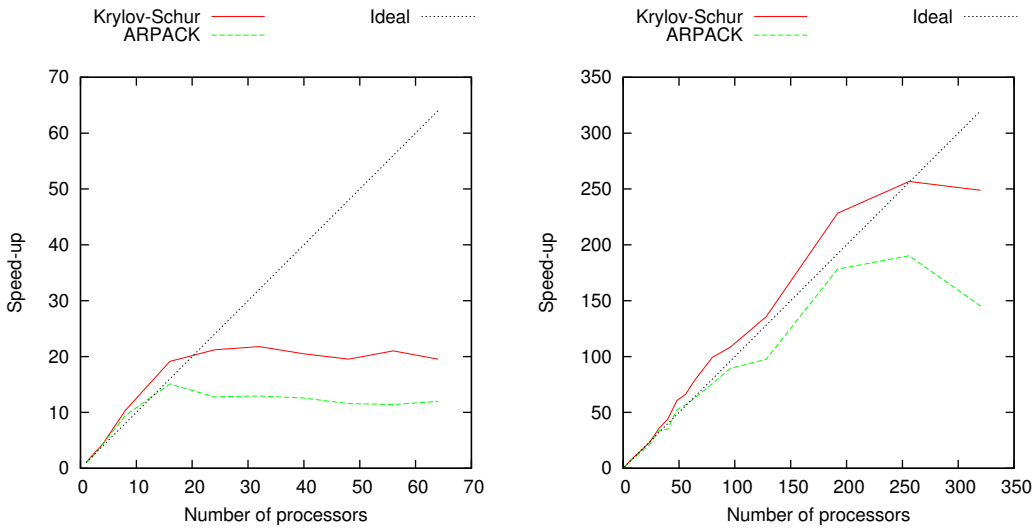

Figure 5: Speed-up in MareNostrum for AF23560 (left) and PRE2 (right) matrices.

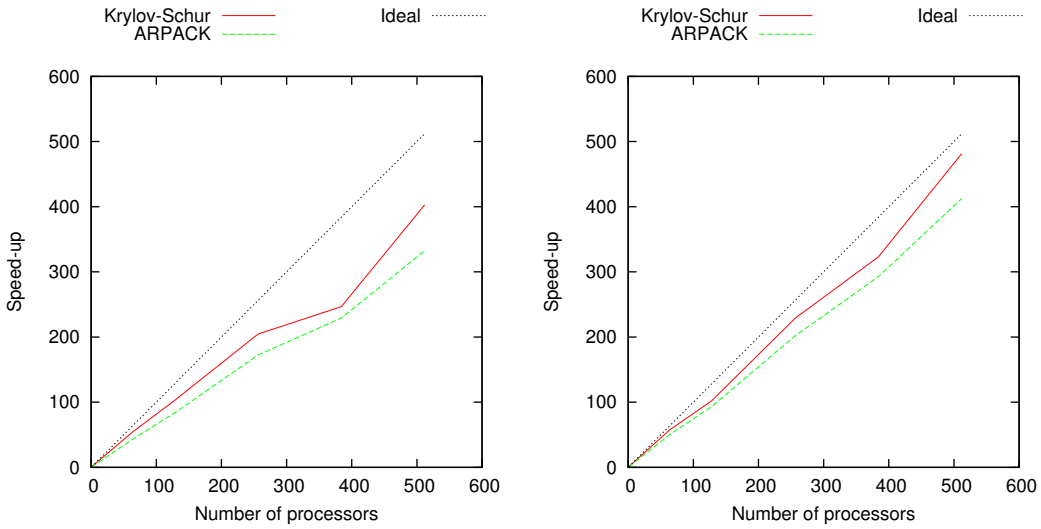

Figure 6: Scaled speed-up in MareNostrum for random tridiagonal matrices with sizes $10,000 \times p$ (left) and $100,000 \times p$ (right).

but using tridiagonal matrices of increasing dimension with random entries. These tridiagonal matrices have 10,000 and 100,000 rows per processor so the amount of computational work remains constant as the number of processors increases. Figure 6 show the scaled speed-ups taking into account the problem size with these two matrices. These tests show the good scalability of the SLEPc and ARPACK implementations, with a slight performance advantage of SLEPc over ARPACK.

References

Bai, Z., D. Day, J. Demmel, and J. Dongarra (1997). A Test Matrix Collection for Non-Hermitian Eigenvalue Problems (Release 1.0). Technical Report CS-97-355, Department of Computer Science, University of Tennessee, Knoxville, TN, USA. Available at http://math.nist.gov/MatrixMarket.

Davis, T. (1992). University of Florida Sparse Matrix Collection. NA Digest. Available at http://www.cise.ufl.edu/research/sparse/matrices.

Kressner, D. (2005). *Numerical Methods for General and Structured Eigenvalue Problems*, volume 46 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin.

Lehoucq, R. B., D. C. Sorensen, and C. Yang (1998). *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Sorensen, D. C. (1992). Implicit Application of Polynomial Filters in a k-Step Arnoldi Method. *SIAM J. Matrix Anal. Appl.*, 13:357–385.

Stewart, G. W. (2001). A Krylov–Schur Algorithm for Large Eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614.

Stewart, G. W. (2002). Addendum to "A Krylov–Schur Algorithm for Large Eigenproblems". *SIAM J. Matrix Anal. Appl.*, 24(2):599–601.

Wu, K. and H. Simon (2000). Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems. *SIAM J. Matrix Anal. Appl.*, 22(2):602–616.

Zhou, Y. and Y. Saad (2004). Block Krylov-Schur Method for Large Symmetric Eigenvalue Problems. Technical Report UMSI-2004-215, Dept. of Computer Science and Eng., University of Minnesota.