Scalable Library for Eigenvalue Problem Computations

SLEPc Technical Report **STR-4**
Available at http://slepc.upv.es

Arnoldi Methods in SLEPc

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

Last update: October, 2006 (SLEPc 2.3.2)
Previous updates: SLEPc 2.3.0

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 method of Arnoldi [1951] was proposed as a means of reducing a matrix to Hessenberg form, but Arnoldi already suggested that the process could provide good approximations to some eigenvalues if stopped before completion. This idea was fully developed in later work by Saad [1980] and other authors, making the method evolve into one of the most successful ones for non-Hermitian eigenproblems.

Many variants of the method have been proposed. This report provides a general description of the algorithm (section 2) and then focuses on the particular variant implemented in SLEPc (section 3).

2 Description of the Method

This section provides an overview of the Arnoldi method and some of its variations. For more detailed background material the reader is referred to [Saad, 1992] or [Bai *et al.*, 2000].

2.1 Basic Arnoldi Algorithm

Given a square matrix A of order n, if n steps of Arnoldi's method are carried out then an orthogonal reduction to Hessenberg form is achieved, $AV = VH$, where H is an upper Hessenberg matrix of order n with positive subdiagonal elements and V is an orthogonal matrix. V and H are uniquely determined by the first column of V, a unit-norm vector $v_1 = Ve_1$ that is called the initial vector. For some initial vectors, Arnoldi's method fails after m steps, and in that case the algorithm produces an $n \times m$ matrix V_m with orthogonal columns and an upper

Hessenberg matrix of order m, H_m, that satisfy

$$AV_m - V_m H_m = 0, \tag{1}$$

that is, the columns of V_m span an invariant subspace of matrix A. In the general case in which the algorithm does not break down, after m steps the following relation holds

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

where vector f is usually called the *residual* of the m-step Arnoldi factorization.

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 $AV_m - V_m H_m = f e_m^*$, $\beta = \|f\|_2$

```
For j = 1, 2, ..., m − 1
    w = Av_j
    Orthogonalize w with respect to V_j (obtaining h_{1:j,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_{1:m,m})
β = ‖f‖_2
```

It can be shown that Algorithm 1 builds an orthogonal basis of the Krylov subspace $\mathcal{K}_m(A, v_1)$. In order to maintain a good level of orthogonality, an iterative Gram-Schmidt procedure must be used for orthogonalization (for details, see SLEPc Technical Report STR-1, "Orthogonalization Routines in SLEPc"). The computed basis vectors are the columns of V_m, which are called Arnoldi vectors. If some w is zero after orthogonalization then the algorithm breaks down before completing m steps and, in that case, the residual of the Arnoldi factorization is zero and $\mathcal{K}_j(A, v_1)$, $j < m$, is an exact invariant subspace of A, as mentioned above. However, this situation is very unlikely in practice due to finite precision arithmetic.

Since $V_m^* f = 0$ by construction, then by premultiplying equation (2) by V_m^*

$$V_m^* A V_m = H_m, \tag{3}$$

that is, matrix H_m represents the orthogonal projection of A onto the Krylov subspace, and this fact allows us to compute Rayleigh-Ritz approximations of the eigenpairs of A. Let (λ_i, y_i) be an eigenpair of matrix H_m, then the Ritz value, λ_i, and the Ritz vector, $x_i = V_m y_i$, can be taken as approximations of an eigenpair of A. Typically, only a small percentage of the m approximations are good. This can be assessed by means of the residual norm for the Ritz pair, which turns out to be very easy to compute:

$$\|Ax_i - \lambda_i x_i\|_2 = \|AV_m y_i - \lambda_i V_m y_i\|_2 = \|(AV_m - V_m H_m) y_i\|_2 = \beta |e_m^* y_i|. \tag{4}$$

In practical situations, the number of steps, m, required to obtain good approximations may be too large. The problem with a large m is not only storage but also the computational cost that grows in every step. For this reason, a restarted version of the algorithm is necessary in practice. The general idea of restarting is that after V_m has been computed, for a fixed value of m, a new Arnoldi process is started, trying to benefit from the previously computed information. In the sequel, m will denote the maximum number of vectors allowed for the basis (`ncv` in the SLEPc terminology).

2.2 Explicit Restart

The idea of explicit restart is to iteratively compute different m-step Arnoldi factorizations with successively "better" initial vectors. The initial vector for the next Arnoldi run is computed from the information available in the most recent factorization. The simplest way to select the new initial vector is to take the Ritz vector (or Schur vector) associated to the dominant eigenvalue, $v_1 = V_m y_1$. This strategy is described below whereas more sophisticated approaches are postponed until next subsection.

In order for a restarted method to be effective in computing more than one eigenpair, it is necessary to keep track of already converged eigenpairs and perform some form of deflation. This is done by a technique usually called *locking*, in which vectors associated to converged eigenvalues are not modified in successive runs. Suppose that after certain Arnoldi run, the first k eigenpairs have already converged to the desired accuracy, and write V_m as

$$V_m = \left[\begin{array}{c|c} V_{1:k}^{(l)} & V_{k+1:m}^{(a)} \end{array} \right], \tag{5}$$

where the (l) superscript indicates locked vectors and the (a) superscript indicates active vectors. In the next Arnoldi run, only $m-k$ Arnoldi vectors must be computed, the active ones, and in doing this the first k vectors have to be deflated. This can be done simply by orthogonalizing every new Arnoldi vector also with respect to the locked ones, as illustrated in Algorithm 2.

Algorithm 2 (Arnoldi with Deflation)

Input: Matrix A, number of steps m, $V_{1:k}$, $H_{1:k}$ with $k < m$, and initial vector v_{k+1} of norm 1
Output: (V_m, H_m, f, β) so that $AV_m - V_m H_m = f e_m^*$, $\beta = \|f\|_2$

For $j = k+1, \ldots, m-1$
  $w = Av_j$
  Orthogonalize w with respect to V_j (obtaining $h_{1:j,j}$)
  $h_{j+1,j} = \|w\|_2$
  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 $h_{1:m,m}$)
$\beta = \|f\|_2$

Note that Algorithm 2 only modifies the last $m-k$ columns of V_m and H_m, the active part of the factorization. All the operations of the algorithm are performed on this active part. These operations are the computation of the Arnoldi factorization with initial vector v_{k+1} and then the computation of the Schur form, see Algorithm 3.

Algorithm 3 (Explicitly Restarted Arnoldi)

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 $V_m = [v_1], \quad k = 0$
Restart loop
  Perform $m-k$ steps of Arnoldi with deflation (Algorithm 2)
  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 eigenvectors of H_m, $H_m y_i = y_i \theta_i$
  Compute residual norm estimates, $\tau_i = \beta |e_m^* y_i|$
  Lock converged eigenpairs
  $V_m \leftarrow V_m U$
end

Algorithm 3 is based on the (real) Schur form instead of the eigendecomposition in order to be more robust. If we partition H_m conforming to (5), then after the computation of the Arnoldi factorization, H_m has the following form

$$H_m = \left[\begin{array}{ccccc|ccccc} \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times & \times & \times & \times \\ \hline & & & & & \times & \times & \times & \times & \times \\ & & & & & \times & \times & \times & \times & \times \\ & & & & & & \times & \times & \times & \times \\ & & & & & & & \times & \times & \times \\ & & & & & & & & \times & \times \end{array}\right], \tag{6}$$

that is, the leading submatrix of order k is upper (quasi-)triangular and the bottom right block is upper Hessenberg. As the algorithm progresses, the lines in (6) descend as soon as new eigenvalues converge. At the end, when the number of converged eigenvalues is sufficient ($k \geq$ `nev`), $V_{1:k}$ and $H_{1:k,1:k}$ constitute an approximate partial Schur decomposition of matrix A. The transformation from Arnoldi vectors to Schur vectors is realized by the operation $V_m = V_m U$, which needs to be performed only on the newly converged vectors and also on the

next initial vector. Note that the initial vector for the next restart is then the Schur vector of the first non-converged eigenvalue. Note also the sorting operation needed to guarantee that eigenvalues always converge in the required order.

2.3 Other Strategies for Restart

In the restarting strategy described above, the Arnoldi algorithm is rerun using as initial vector the approximate Schur vector associated to the first not-yet-converged eigenvalue. This has the effect that in the next Arnoldi run that particular eigenvalue will converge fast because the initial vector already points in a direction close to the corresponding eigenvector. (Note that this is not the case if the eigenvalue is complex and the algorithm runs in real arithmetic.) However, when several eigenpairs have to be computed, this approach is not the best one, because the other nearly converged eigenpairs are not represented in the initial vector. So a better approach would be to set the initial vector as a linear combination of all the eigenvectors (or Schur vectors). But in doing this, we might be including some directions which are not of interest, so care must be taken to remove from the initial vector the directions associated to unwanted eigenvalues. This more elaborate strategy is usually called *filtering* since in the initial vector the directions corresponding to wanted eigenvalues are amplified whereas those corresponding to unwanted ones are damped.

Polynomial filtering. One possibility for achieving the filtering effect is to take the initial vector to be $v_{k+1} = z/\|z\|_2$ where $z = p(A)z_0$, being p a polynomial of degree d and z_0 a linear combination of the approximate eigenvectors (or Schur vectors). As stated above, this polynomial has to be defined in such a way that the new initial vector is rich in the directions of wanted eigenvectors while other directions are removed. This goal can be achieved by adapting the optimality properties of Chebyshev polynomials [Saad, 1984; Sadkane, 1993]. In this approach, p is a Chebyshev polynomial of the first kind of degree d, $T_d[(A-\sigma I)/\rho]$, where σ and ρ give a translation and scaling of the part of the spectrum one wants to suppress. In the case of non-Hermitian problems, this part of the spectrum is represented by an ellipse containing all the unwanted eigenvalues. Since these eigenvalues are unknown and the optimal ellipse is difficult to define, other alternative approaches have been proposed based on other types of polynomials, such as least-squares polynomials, [Saad, 1987], or Faber polynomials, [Heuveline and Sadkane, 1997].

Polynomial filtering as described here has several drawbacks. One of them is that it is difficult, in general, to choose a linear combination z_0 of the eigenvectors that leads to balanced convergence because it is hard to represent a whole subspace by a single vector. A cure for this could be to apply the filtering in all the steps of the Arnoldi method and not just at the restart, that is, replace matrix A by $p(A)$. This is usually called *polynomial preconditioning*. Another disadvantage is the wide variation in performance depending on the choice of the parameters of the polynomial, making it difficult to implement a robust solver that works well in many situations.

Implicit restart. The restarting schemes discussed so far are explicit in the sense that an initial vector is computed and then a new Arnoldi process is started from scratch. 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 and numerically stable 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]. This technique has been implemented in the ARPACK software, [Lehoucq *et al.*, 1998].

The implicit QR approach is generally superior to explicit restart, because the effect is that the subspace generated by the algorithm contains the Krylov subspace associated to *all* the approximate Schur vectors and not just one. For a more detailed discussion see [Morgan, 1996].

Krylov-Schur. Implementing the implicit restart technique in a numerically stable way is difficult. A simpler way of achieving the same effect is by the so-called Krylov-Schur method, [Stewart, 2001]. Since this method is also implemented in SLEPc, its description is provided separately (see SLEPc Technical Report STR-7, "Krylov-Schur Methods in SLEPc").

2.4 Other Variants

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

B-Arnoldi. When addressing a generalized eigenvalue problem, $Ax = \lambda Bx$, usually a spectral transformation is used. In this case, it is sometimes useful to replace the standard Hermitian inner product, $\langle x, y\rangle = y^*x$, by the B-inner product, $\langle x, y\rangle_B = y^*Bx$. In the context of Algorithm 2, this can be accomplished by doing B-orthogonalization and replacing the 2-norm with the B-norm, $\|w\|_B = \sqrt{\langle w, w\rangle_B}$. This modified algorithm is usually referred to as the B-Arnoldi process.

In the case of Hermitian positive-definite pencils, symmetry is recovered in the transformed operator, e.g. $(A - \sigma B)^{-1}B$, when the B-inner product is used. Of course, this property is relevant only for solvers that exploit symmetry, such as Lanczos (see SLEPc Technical Report STR-5, "Lanczos Methods in SLEPc"). However, the B-Arnoldi process is still useful in the case that B is singular, even when A is not symmetric. If B is singular, then the bilinear form $\langle x, y\rangle_B$ is a semi-inner product, but the B-Arnoldi process is still well defined and can provide a solution to the problem of eigenvector corruption due to infinite eigenvalues, see [Meerbergen and Spence, 1997].

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

$$\mathcal{K}_m(A, V_1) = \text{span}\{V_1, AV_1, A^2V_1, \ldots, A^{m-1}V_1\} \quad , \tag{7}$$

where V_1 has b columns. This kind of method has two main advantages. First, the convergence behavior can be less problematic in cases with multiple or clustered eigenvalues, provided that b is sufficiently large. Second, computational efficiency may be better in some situations, because access to the elements of matrix A is amortized by multiplying several vectors and not just one. On the other hand, one drawback is that in order for the algorithm to work with polynomials of degree m, the required dimension of the subspace is $m \cdot b$, which can be quite large. Another drawback is that issues such as deflation, which are relatively simple to implement in unblocked algorithms, become severe complications in the block variants.

Two approaches are generally possible for developing the above ideas. For distinguishing them, they are sometimes referred to as block and band versions, respectively. The first one is a straightforward block generalization of the simple algorithm, using block Gram-Schmidt orthogonalization. In this case, matrix H_m is no longer Hessenberg but block Hessenberg. In the other alternative, vectors are orthogonalized one at a time, with the usual Gram-Schmidt procedure, resulting in a band Hessenberg matrix.

2.5 Available Implementations

ARNCHEB [Braconnier, 1993] is a Fortran software that implements the Arnoldi method with explicit restart, via the Chebyshev polynomial technique described in section 2.3.

As mentioned above, ARPACK [Lehoucq *et al.*, 1998] provides a Fortran implementation of the Implicitly Restarted Arnoldi method, for both real and complex arithmetic. The software can be downloaded from `http://www.caam.rice.edu/software/ARPACK`. A parallel version is also available, based on BLACS or MPI message passing.

3 The SLEPc Implementation

The variant of the Arnoldi method provided by SLEPc is based on explicit restart with locking. The corresponding solver is `EPSARNOLDI` (or `-eps_type arnoldi` from the command-line).

The algorithm actually implemented is Algorithm 3. Currently, the only restarting strategy available is the simplest one of those described above, that is, the new initial vector is the Schur vector of the first eigenvalue that has not converged yet. Krylov-Schur restart is implemented in SLEPc as a separate eigensolver (see SLEPc Technical Report STR-7, "Krylov-Schur Methods in SLEPc"). Implicit restart is indirectly available in SLEPc via wrapper code to interface to ARPACK (solver `EPSARPACK`, see the SLEPc Users Manual for details on how to enable external software).

Currently, block versions of the algorithms have not been considered in SLEPc. One of the reasons is the limited support for multi-vectors in PETSc.

3.1 User Options

The only parameters that can be adjusted by the user are those related to the orthogonalization technique used e.g. in the third line of Algorithm 2. Apart from the general orthogonalization

options described in SLEPc Technical Report STR-1 ("Orthogonalization Routines in SLEPc"), a specific option is available for the Arnoldi eigensolver, namely the activation of "delayed" variants, which is a rather advanced feature.

```
EPSArnoldiSetDelayed(EPS eps,PetscTruth delayed)
```

This function activates (or deactivates) delayed Arnoldi variants, which are off by default. Delayed variants can also be activated with the command-line option `-eps_arnoldi_delayed`.

The delayed variants introduce either delayed normalization or delayed refinement. These are somewhat aggressive optimizations than may provide better scalability, but sometimes make the solver converge less than the default algorithm. If the delayed flag is activated with the above function, then either delayed normalization (if `refinement` is equal to `never`) or delayed refinement (if `refinement` is equal to `always`) is carried out (see SLEPc Technical Report STR-1 for instructions on how to change the parameter `refinement`).

Full details about delayed variants, including numerical results and performance analysis, can be found in [Hernandez *et al.*, 2007].

3.2 Known Issues and Applicability

Arnoldi is probably the most robust SLEPc eigensolver. However, its simple restart mechanism makes it converge generally slower compared to other solvers such as Krylov-Schur.

Large errors may appear in the computed eigenvectors in the case of a generalized eigenproblem with singular B.

With respect to delayed variants, Arnoldi with delayed refinement may suffer from convergence stagnation in some cases. For this reason, it is not used by default.

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

References

Arnoldi, W. E. (1951). The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem. *Quart. Appl. Math.*, 9:17–29.

Bai, Z., J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst (eds.) (2000). *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Braconnier, T. (1993). The Arnoldi-Tchebycheff Algorithm for Solving Large Nonsymmetric Eigenproblems. Technical Report TR/PA/93/25, CERFACS, Toulouse, France.

Hernandez, V., J. E. Roman, and A. Tomas (2007). Parallel Arnoldi Eigensolvers with Enhanced Scalability via Global Communications Rearrangement. *Parallel Comput.*, 33(7–8):521–540.

Heuveline, V. and M. Sadkane (1997). Arnoldi-Faber Method for Large non Hermitian Eigenvalue Problems. *Electron. Trans. Numer. Anal.*, 5:62–76.

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.

Meerbergen, K. and A. Spence (1997). Implicitly Restarted Arnoldi with Purification for the Shift-Invert Transformation. *Math. Comp.*, 66(218):667–689.

Morgan, R. B. (1996). On Restarting the Arnoldi Method for Large Nonsymmetric Eigenvalue Problems. *Math. Comp.*, 65:1213–1230.

Saad, Y. (1980). Variations of Arnoldi's Method for Computing Eigenelements of Large Unsymmetric Matrices. *Linear Algebra Appl.*, 34:269–295.

Saad, Y. (1984). Chebyshev Acceleration Techniques for Solving Nonsymmetric Eigenvalue Problems. *Math. Comp.*, 42:567–588.

Saad, Y. (1987). Least Squares Polynomials in the Complex Plane and their Use for Solving Nonsymmetric Linear Systems. *SIAM J. Numer. Anal.*, 24(1):155–169.

Saad, Y. (1992). *Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms*. John Wiley and Sons, New York.

Sadkane, M. (1993). A Block Arnoldi-Chebyshev Method for Computing Leading Eigenpairs of Large Sparse Unsymmetric Matrices. *Numer. Math.*, 64:181–194.

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.