Romero Alcalde, E.; Román Moltó, JE. (2014). A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc. ACM Transactions on Mathematical Software. 40(2):13:01-13:29. doi:10.1145/2543696.

# A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc

ELOY ROMERO and JOSE E. ROMAN, Universitat Politècnica de València

In the context of large-scale eigenvalue problems, methods of Davidson type such as Jacobi-Davidson can be competitive with respect to other types of algorithms, especially in some particularly difficult situations such as computing interior eigenvalues or when matrix factorization is prohibitive or highly inefficient. However, these types of methods are not generally available in the form of high-quality parallel implementations, especially for the case of non-Hermitian eigenproblems. We present our implementation of various Davidson-type methods in SLEPc, the Scalable Library for Eigenvalue Problem Computations. The solvers incorporate many algorithmic variants for subspace expansion and extraction, and cover a wide range of eigenproblems including standard and generalized, Hermitian and non-Hermitian, with either real or complex arithmetic. We provide performance results on a large battery of test problems.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematical Software**]

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: eigenvalue computations, Davidson, Jacobi-Davidson, SLEPc, message-passing parallelization

**ACM Reference Format:**
Romero, E., and Roman, J. E. 2012. A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc. ACM Trans. Math. Softw. 1, 1, Article 11 (January 1111), 29 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

## 1. INTRODUCTION

SLEPc, the Scalable Library for Eigenvalue Problem Computations [Hernandez et al. 2005], is a parallel library that provides state-of-the-art algorithms and tools to solve large-scale eigenvalue problems, including linear eigenproblems as well as other related problems like singular value decompositions and quadratic eigenproblems. These problems appear frequently in different scientific disciplines, like for instance nuclear engineering, electromagnetics and electronic structure calculations, and demand large computational effort. In this context, it is necessary to make use of high-end computing platforms, such as clusters of computers, and, ever more, emerging hybrid architectures that combine multi-cores and accelerators.

In this paper we focus on the (linear) generalized eigenvalue problem, that is, the computation of eigenvalue-eigenvector pairs $(\lambda_i, \mathbf{x}_i)$ satisfying the equation $A\mathbf{x}_i = \lambda_i B\mathbf{x}_i$, where $A$ and $B$ are square matrices, either real or complex. The case $B = I$ is often referred to as the standard eigenvalue problem. We will devote especial attention

This work was supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2009-07519.
Author's addresses: E. Romero, Institut I3M, Universitat Politècnica de València, Camí de Vera s/n, 46022 Valencia (Spain), and J. E. Roman, Departament de Sistemes Informátics i Computació, Universitat Politècnica de València, Camí de Vera s/n, 46022 Valencia (Spain).

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Table I. Summary of parallel libraries providing basic methods (subspace iteration, RQI and LOBPCG) as well as Davidson-type solvers, with eigenproblem types supported by each method (HEP: standard Hermitian, GHEP: generalized Hermitian, GNHEP: generalized non-Hermitian).

| | | **Libraries** | | | |
|---|---|---|---|---|---|
| | | PRIMME | Anasazi | SLEPc | BLOPEX |
| **Methods** | Subspace Iteration | HEP | | GNHEP | |
| | Rayleigh Quotient Iteration | HEP | | GNHEP | |
| | LOBPCG | HEP | GHEP | | GHEP |
| | Generalized Davidson | HEP | GHEP | GNHEP | |
| | Jacobi-Davidson | HEP | GHEP* | GNHEP | |

*Anasazi requires the user to implement the Jacobi-Davidson correction equation.

to symmetric (Hermitian) problems, where both $A$ and $B$ are symmetric (Hermitian) and $B$ is positive (semi-)definite. Singular matrix pairs $(A, B)$, that is, when $A$ and $B$ have a common null space, are not considered in this work. We are concerned with applications where $A$ and $B$ are large and sparse, and only a small percentage of the eigenvalues are required. The methods for this are iterative in nature, and compute approximate eigenpairs $(\theta_i, \tilde{\mathbf{x}}_i)$ such that the residual $\|A\tilde{\mathbf{x}}_i - \theta_i B\tilde{\mathbf{x}}_i\|$ is bounded by a given tolerance.

Among the methods available for addressing the above problem we can find: (i) classical methods such as the power iteration, subspace iteration and Rayleigh quotient iteration (RQI); (ii) Krylov methods such as Arnoldi, Lanczos and Krylov-Schur, that are particularly suitable for computing extreme eigenvalues; (iii) preconditioned conjugate gradient methods such as LOBPCG [Knyazev 2001], that is efficient and robust for generalized symmetric eigenproblems; and (iv) Davidson methods such as Generalized Davidson and Jacobi-Davidson, that prove being very efficient compared with the rest of the methods when computing eigenvalues located in the interior of the spectrum and/or when the problem is generalized ($B \neq I$). See [Bai et al. 2000; Stewart 2001; van der Vorst 2002] for a more detailed overview of these methods.

Krylov methods are very popular for computing eigenvalues in the periphery of the spectrum, and are available in the form of parallel implementations such as ARPACK [Lehoucq et al. 1998]. The main drawback of these methods is that they often require to implicitly handle a matrix inverse, *e.g.*, when computing interior eigenvalues with the shift-and-invert technique [Ericsson and Ruhe 1980]. This implies solving linear systems at each eigensolver iteration, and for this most often direct solvers must be employed to guarantee robustness. Methods belonging to the Davidson family try to overcome this limitation by relaxing the precision with which these inverses are approximated (in some cases with a simple preconditioner). As opposed to Krylov methods, general-purpose Davidson solvers are still difficult to find in freely available parallel software, especially with capabilities for non-symmetric and/or generalized problems, despite there being numerous publications developing the methods for different problem types (as §2 summarizes) and even describing parallel implementations tailored for certain applications [Heuveline et al. 1997; Nool and van der Ploeg 2000; Arbenz et al. 2006; Genseberger 2010; Hwang et al. 2010; Ferronato et al. 2012].

The main parallel libraries that provide eigensolvers similar to ours are summarized in Table I: PRIMME [Stathopoulos and McCombs 2010], that includes many methods that fit in the Generalized Davidson scheme, although currently without support for generalized problems; Anasazi [Baker et al. 2009], that includes a customizable block Generalized Davidson (that with a little coding can behave like a basic Jacobi-

Davidson) and LOBPCG; and BLOPEX[1] [Knyazev et al. 2007], that provides a reference implementation of LOBPCG. As Table I shows, none of the mentioned libraries support non-Hermitian problems.

This paper presents our developments in SLEPc in order to provide state-of-the-art, robust, high performance Davidson solvers supporting Hermitian and non-Hermitian eigenproblems, both standard and generalized, with either real or complex arithmetic.

In §2 we briefly summarize the theoretical background of the implemented Davidson methods and their variants. Section 3 starts with an overview of SLEPc and its foundation PETSc (Portable, Extensible Toolkit for Scientific Computation [Balay et al. 1997]), followed by an outline on the design of the Davidson solvers and a description of their user interfaces. Then in §4 we discuss how to implement some non-trivial aspects described in previous sections. In §5 we present sequential and parallel performance results of our implementation, with a number of test problems as well as a few real applications. Finally, we end with the conclusions.

## 2. DAVIDSON-TYPE FRAMEWORK

We present a review of the Davidson methods and related techniques that we have implemented in SLEPc. For an extensive bibliography about Davidson methods see [Bai et al. 2000; van der Vorst 2002; 2004; Hochstenbach and Notay 2006]. First we describe the main steps of this type of methods, and later subsections detail the variants of each step that are available in the implementation. Unfortunately, often there is no mathematical proof that indicates the best way of carrying out each step in the general case, that is, the optimal configuration is problem dependent. For that reason, it is customary that libraries offering Davidson methods provide several variants for each step and mechanisms to customize the execution.

### 2.1. General Description of the Davidson-type Methods

Subspace methods seek the eigenvectors in a low-dimensional search subspace, which is updated at every iteration. Davidson-type methods are a distinguished subclass of the subspace methods that expand the search subspace $\mathcal{V}$ in the directions of the computed corrections to the most wanted eigenvectors in the search subspace. In fact a Davidson variant is characterized by how to select the wanted eigenpairs in the search subspace (*extraction*) and how to compute the corrections (*expansion*). The convergence of the eigenpairs is tracked, for instance monitoring the norm of the residual associated with the approximate eigenpair, $\mathbf{r}_i$, or its correction, $\mathbf{d}_i$. When an eigenpair is considered converged it is removed from the search subspace and a *deflation* technique is used in order to prevent the convergence of the same pair afterward. When the dimension of the search subspace $\mathcal{V}$ grows up to a certain limit $m_{\max}$, it is reset to an $m_{\min}$-dimensional subspace $\mathcal{V}' \subset \mathcal{V}$ keeping as much useful spectral information as possible (*restart*). Algorithm 1 provides a general scheme of a Davidson-type method. We next discuss some general issues, and postpone the details of each step until later subsections, where the theoretical background of the different alternatives is introduced.

From the numerical point of view, working with an orthogonal basis of the search subspace $V$ is desirable to control numerical error and also to maintain the whole set of vectors linearly independent after the addition of the correction vectors $D$. The extraction of approximations is based on a projection on this subspace, and it is also desirable to take into account any kind of structure (e.g., symmetry) present in the

[1]LOBPCG is actually not a Davidson-type method, but it has a similar scheme, and GD with restarting strategy GD$(b, 3b)+b$ is mathematically equivalent to LOBPCG with block size $b$, see §2.5 and [Stathopoulos and McCombs 2010].

**ALGORITHM 1:** Basic Davidson-type Method

**Input**: matrices $A$ and $B$ of size $n$, number of wanted eigenpairs $p$, block size $s$, initial dimension of $V$ $m_0$, maximum dimension of $V$ $m_{\max}$, restart with $m_{\min}$ vectors
**Output**: resulting eigenpairs $(\Theta, X)$

Choose a starting subspace basis $V$ of $m_0$ vectors
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow [\,]$ and $X \leftarrow [\,]$
**while** $l < p$ **do**
  Extraction: Compute the Ritz pairs $(\widetilde{\Theta}, \widetilde{X})$ from the projected eigenproblem and sort them
  Test convergence, store the $k$ converged pairs in $(\Theta, X)$ and remove them from $(\widetilde{\Theta}, \widetilde{X})$; set $m \leftarrow m - k$ and $l \leftarrow l + k$
  **if** $m \geq m_{\max}$ **then** Restart $V$ with an $m_{\min}$-dimensional subspace basis and set $m \leftarrow m_{\min}$
  Expansion: Compute the correction $D$ of the first $s$ pairs $(\widetilde{\Theta}, \widetilde{X})$, and add them to $V$; set $m \leftarrow m + s$
**end**

**ALGORITHM 2:** Generalized Hermitian Davidson-type Method with $B$-orthogonalization

**Input**: matrices $A$ and $B$ of size $n$, preconditioner $K$, number of wanted eigenpairs $p$, block size $s$, initial dimension of $V$ $m_0$, maximum size of $V$ $m_{\max}$, restart with $m_{\min}$ vectors
**Output**: resulting eigenpairs $(\Theta, X)$

Choose a starting subspace basis $V$ of $m_0$ vectors, such that $V^*BV = I_{m_0}$
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow [\,]$ and $X \leftarrow [\,]$
**while** $l < p$ **do**
  Extraction: Compute the Ritz pairs $(\widetilde{\Theta}, \widetilde{X})$ by means of the Rayleigh-Ritz method, that is, solve $V^*AVU = U\widetilde{\Theta}$, where $U^*U = I_m$ and $\widetilde{X} = VU$
  Sort the Ritz pairs $(\widetilde{\Theta}, \widetilde{X})$
  Obtain the number of converged pairs $k$
  **if** $k > 0$ **then**
    Add eigenvalues $\widetilde{\theta}_1, \ldots, \widetilde{\theta}_k$ to $\Theta$
    Set $X \leftarrow [X \quad \widetilde{X}_{1:k}]$ and $V \leftarrow \widetilde{X}_{k+1:m}$
    Set $m \leftarrow m - k$ and $l \leftarrow l + k$
  **end**
  **if** $m \geq m_{\max}$ **then**
    Choose an $m \times m_{\min}$-matrix $M$ to reset $V$, $V \leftarrow VM$, such that $M^*M = I_{m_{\min}}$
    Update $(\widetilde{\Theta}, \widetilde{X})$ and $U$ so that $V^*AVU = U\widetilde{\Theta}$, where $U^*U = I_{m_{\min}}$ and $\widetilde{X} = VU$
    Set $m \leftarrow m_{\min}$
  **end**
  Expansion: Compute the correction $D$ of the first $s$ pairs $(\widetilde{\Theta}, \widetilde{X})$
  $V \leftarrow [V \quad B\text{-orthonormalize}([X \quad V], D)]$ and set $m \leftarrow m + s$
**end**

original problem in such a way that this structure is preserved in the projected problem. This has implications on how the Davidson method is realized. Therefore, the general scheme is specialized depending on the properties of the eigenproblem.

For instance, in a standard Hermitian or generalized Hermitian-definite problem (with $B$ positive definite) we know that eigenvectors are $B$-orthogonal, so keeping a $B$-orthogonal basis $V$ will result in a standard Hermitian problem coming out from the projection, if the Rayleigh-Ritz procedure is used. The deflation is performed by $B$-orthogonalizing the new vectors $D$ against the previously converged eigenvectors $X$. This variant is detailed in Algorithm 2.

**ALGORITHM 3:** Generalized non-Hermitian Davidson-type Method

**Input**: matrices $A$ and $B$ of size $n$, preconditioner $K$, number of wanted eigenpairs $p$, block size $s$, initial dimension of $V$ $m_0$, maximum size of $V$ $m_{\max}$, restart with $m_{\min}$ vectors
**Output**: resulting eigenvalues $\Theta$ and Schur vectors $X$

Choose a starting subspace basis $V$ of $m_0$ vectors, such that $V^*V = I_{m_0}$
Compute $W$ corresponding to $V$, such that $W^*W = I_{m_0}$
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow [\,]$, $X \leftarrow [\,]$ and $Y \leftarrow [\,]$
**while** $l < p$ **do**
  Extraction: Compute the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$, from the Generalized Schur decomposition $W^*AV = ZSU^*$ and $W^*BV = ZTU^*$, where $\widetilde{X} = VU$ and $\widetilde{\theta}_i = s_{i,i}/t_{i,i}$
  Sort the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$
  Obtain the number of converged pairs $k$
  **if** $k > 0$ **then**
    Add eigenvalues $\widetilde{\theta}_1, \ldots, \widetilde{\theta}_k$ to $\Theta$
    Set $X \leftarrow [X \quad \widetilde{X}_{1:k}]$, $Y \leftarrow [Y \quad WZ_{1:k}]$
    Set $V \leftarrow VU_{k+1:m}$ and $W \leftarrow [W \quad WZ_{k+1:m}]$
    Set $m \leftarrow m - k$ and $l \leftarrow l + k$
  **end**
  **if** $m \geq m_{\max}$ **then**
    Choose an $m \times m_{\min}$-matrix $M$ to reset $V$, $V \leftarrow VM$, such that $M^*M = I_{m_{\min}}$
    Update $(\widetilde{\Theta}, \widetilde{X})$, $U$ and $Z$ so that $W^*AV = ZSU^*$ and $W^*BV = ZTU^*$, where $\widetilde{X} = VU$ and $\widetilde{\theta}_i = s_{i,i}/t_{i,i}$
    Set $m \leftarrow m_{\min}$
  **end**
  Expansion: Compute the correction $D$ of the first $s$ pairs $(\widetilde{\Theta}, \widetilde{X})$
  Set $V \leftarrow [V \quad \text{orthonormalize}([X \quad V], D)]$
  Compute $W^0$ corresponding to $V_{m:m+s}$ and set $W \leftarrow [W \quad \text{orthonormalize}([Y \quad W], W^0)]$
  Set $m \leftarrow m + s$
**end**

Algorithm 2 returns eigenvectors with unit $B$-norm. However, if $B$ is numerically singular, *i.e.*, $|\mathbf{x}^*B\mathbf{x}|$ is close to zero for some eigenvector $\mathbf{x}$ (that is, an eigenvector corresponding to an infinite eigenvalue of the matrix pair $(A, B)$), enforcing $B$-normality may result in breakdown if a Ritz vector converges to $\mathbf{x}$. If $\alpha \in \mathbb{R}$ can be found such that $A - \alpha B$ is nonsingular, a straightforward solution would be to run Algorithm 2 on the matrix pair $(A - \beta B, A - \alpha B)$, that has the same eigenvectors as the original problem and the eigenvalues

$$\tilde{\mu}_i = \frac{\tilde{\theta}_i - \beta}{\tilde{\theta}_i - \alpha}. \tag{1}$$

When the problem is not Hermitian, the implemented Davidson-type method works completely with generalized Schur decompositions (see Algorithm 3), that for a matrix pair $(\hat{A}, \hat{B})$ is the decomposition $\hat{A}U = ZS$ and $\hat{B}U = ZT$, where $U$ and $Z$ are unitary matrices and $S$ and $T$ are upper triangular matrices.

The eigenvectors of non-Hermitian eigenproblems do not have to form an orthonormal set (nor $B$-orthonormal), so an algorithm that operates directly with them would be dangerous from the numerical point of view. In contrast, Schur decompositions are numerically more stable because of the use of orthonormal bases of invariant subspaces, which also simplifies the task of deflation. Schur forms have the additional

benefit of using quotients $s_{i,i}/t_{i,i}$ for representing the eigenvalues, thus handling infinite eigenvalues more naturally with $t_{i,i} = 0$.

Moreover, an additional advantage for the case of real non-symmetric eigenproblems is the use of real Schur forms, which work with real orthogonal bases of the invariant subspaces associated with the eigenvectors, that may be complex, hence avoiding complex arithmetic completely. In this form $S$ and $T$ are upper quasi-triangular, possibly with $2 \times 2$ diagonal blocks representing complex conjugate pairs of eigenvalues.

In [Fokkema et al. 1998] a Jacobi-Davidson method using generalized Schur forms is introduced (called JDQZ), whose main difference with respect to the version presented in this work is that vectors used for deflation of the search subspace (see §2.4) must be necessarily included in the projectors in the correction equation (see §2.3). In our implementation this is optional and, if necessary, deflation is completed when a new vector is added to the bases $V$ and $W$.

Consider the partial generalized Schur decomposition $AX = YS$ and $BX = YT$ corresponding to already converged eigenpairs. The method considers that the Schur tuple $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, (\alpha, \beta))$, which comes from the Schur form of the projected problem in the extraction, has converged and must be appended to the decomposition, that is,

$$A\begin{bmatrix} X & \tilde{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} Y & \tilde{\mathbf{y}} \end{bmatrix} \begin{bmatrix} S & \mathbf{s} \\ 0 & \sigma \end{bmatrix} \text{ and } B\begin{bmatrix} X & \tilde{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} Y & \tilde{\mathbf{y}} \end{bmatrix} \begin{bmatrix} T & \mathbf{t} \\ 0 & \tau \end{bmatrix}. \tag{2}$$

So the only condition that the new tuple should satisfy is $X^*\tilde{\mathbf{x}} = Y^*\mathbf{y} = 0$ or, equivalently, that $X^*V = Y^*W = 0$, because $\tilde{\mathbf{x}} \in \text{span}\{V\}$ and $\tilde{\mathbf{y}} \in \text{span}\{W\}$. In the implementation this condition is enforced by maintaining the bases $V$ and $W$ orthogonal against $X$ and $Y$, respectively. This variant is detailed in Algorithm 3.

Both Algorithm 2 and 3 support computing the correction of more than one approximate eigenpair, by increasing the value of parameter $s$ (block size). However, in terms of convergence our experience shows that the optimal value for $s$ is 1 (the same comment appears in [Stathopoulos and McCombs 2007, §2.2.2]).

In step 5 of Algorithm 3 the generalized Schur decomposition is sorted so that the wanted pairs are located at the beginning of the decomposition (see [Kressner 2006] and references therein). The first $s$ pairs will be corrected at the next steps, and when restarting, the first $m_{\min}$ pairs determine the part preserved in $V$ and $W$.

We conclude this subsection with a brief comment about the case of Hermitian-indefinite problems. When both $A$ and $B$ are Hermitian matrices, but $B$ is indefinite, then Algorithm 2 cannot be used but it is still possible to exploit symmetry to some extent. If $B^{-1}A$ is non-defective, the eigenvectors $X$ satisfy $X^TBX = I^{\pm}$, where $I^{\pm}$ is a signature matrix (diagonal with $\pm 1$ elements on the diagonal). For this case, we have implemented a variant of Algorithm 3 where $W = V$, the orthogonalization is performed with respect to the $B$ inner product (that is an indefinite inner product or a pseudo-inner product), and the resulting vectors are normalized so that $|\mathbf{x}^TB\mathbf{x}| = 1$. The corresponding projected problem is a generalized symmetric-indefinite eigenproblem and can be solved with a structure-preserving method like the one proposed in [Brebner and Grad 1982]. However, since this solver is not available as a LAPACK subroutine, we currently use the Schur decomposition instead.

### 2.2. Subspace Extractions

The extraction techniques considered in this section return a set of approximate eigenpairs $(\tilde{\theta}, \tilde{\mathbf{x}})$ whose vectors belong to the search subspace $\mathcal{V}$ spanned by the columns of $V$, that is, $\tilde{\mathbf{x}} = V\mathbf{u}$. The Rayleigh-Ritz approach is the most basic one and is considered useful for computing eigenvalues at the periphery of the spectrum (see [Stewart 2001, §4.4]). This technique imposes the Ritz-Galerkin condition on the residual associated

with the returned pair,

$$\mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}B\tilde{\mathbf{x}} \perp \mathcal{V}, \tag{3}$$

which leads to the low-dimensional projected eigenproblem

$$V^*AV\mathbf{u} = \tilde{\theta}V^*BV\mathbf{u}. \tag{4}$$

From the solutions $(\tilde{\theta}, \mathbf{u})$ of (4), the returned pairs $(\tilde{\theta}, \tilde{\mathbf{x}}) = (\tilde{\theta}, V\mathbf{u})$ are obtained, which are called Ritz pairs.

The projected matrices $V^*AV$ and $V^*BV$ preserve the Hermitian structure of the problem matrices $A$ and $B$, and in case the search subspace basis $V$ is $B$-orthogonal, (4) will be a standard Hermitian eigenproblem, as indicated in line 4 of Algorithm 2.

However, the Rayleigh-Ritz approach generally returns poor approximate eigenvectors for interior eigenvalues (see, for instance, [Stewart 2001, Example 4.2]). A simple explanation for that is that the Ritz-Galerkin condition (3) does not consider the residual norm of the resulting eigenpairs (for a more detailed explanation see [Sleijpen et al. 1998, §4]). The harmonic Rayleigh-Ritz method [Morgan 1991; Paige et al. 1995] was proposed instead as an alternative technique for interior eigenvalues, that imposes the Petrov-Galerkin condition to the residual of the returned pairs $(\tilde{\theta}, \tilde{\mathbf{x}})$

$$A\tilde{\mathbf{x}} - \tilde{\theta}B\tilde{\mathbf{x}} \perp \mathcal{W} := (A - \tau B)\mathcal{V}, \tag{5}$$

if eigenvalues close to $\tau$ are sought. Similarly to the previous case, this leads to the projected eigenproblem

$$V^*(A - \tau B)^*(A - \tau B)V\mathbf{u} = \xi V^*(A - \tau B)^*BV\mathbf{u}. \tag{6}$$

The solution pairs $(\xi, \mathbf{u})$ of (6) with smallest $\xi$ correspond to the harmonic Ritz pairs $(\tilde{\theta}, \tilde{\mathbf{x}}) = (\tau + \xi, V\mathbf{u})$ closest to the target $\tau$, which satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq |\xi|\|B\tilde{\mathbf{x}}\|, \tag{7}$$

resulting from left-multiplying (6) by $\mathbf{u}^*$ and applying the Cauchy-Schwarz inequality. Hence it is sensible to think (at least for eigenvalues sufficiently close to $\tau$) that selecting the pairs $(\tilde{\theta}, \tilde{\mathbf{x}})$ with $\tilde{\theta}$ closest to $\tau$ (*i.e.*, with smallest $|\xi|$) also correspond to the pairs with smallest residual norm.

Recently, two new variations of the harmonic extraction have been proposed. One is the relative harmonic extraction [Hochstenbach 2005b] that finds eigenvalues $\theta$ with minimal

$$|\tilde{\theta} - \tau||\tilde{\theta}|^{-1} = |1 - \tau\tilde{\theta}^{-1}|, \tag{8}$$

that is, eigenvalues $\tilde{\theta}$ closest to $\tau$ considering the *weight* of $\tilde{\theta}$, in contrast to harmonic extraction. These approximate eigenpairs $(\tilde{\theta}, \mathbf{x})$ can be obtained by the constraint

$$A\tilde{\mathbf{x}} - \tau(1 - \tau\tilde{\theta}^{-1})^{-1}(A - \tau B)\tilde{\mathbf{x}} \perp \mathcal{W}. \tag{9}$$

The resulting eigenpairs satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq |1 - \tau\tilde{\theta}^{-1}|\|A\tilde{\mathbf{x}}\|. \tag{10}$$

The other one is specific for extracting the rightmost eigenvalues [Hochstenbach 2005a], particularly when eigenvalues with large imaginary part are present in the spectrum. These eigenpairs are selected with the Galerkin condition,

$$(A + \bar{\tau}B)\tilde{\mathbf{x}} - \frac{\tilde{\theta} + \bar{\tau}}{\tilde{\theta} - \tau}(A - \tau B)\tilde{\mathbf{x}} \perp \mathcal{W}, \tag{11}$$

Table II. Correspondence between the values of $\alpha$, $\beta$, $\gamma$ and $\delta$ in the generic Galerkin condition (13) and some extraction methods.

| Extraction | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ |
|---|---|---|---|---|
| Harmonic Rayleigh-Ritz | 1 | $\tau$ | 0 | 1 |
| Relative harmonic Rayleigh-Ritz | 1 | $\tau$ | 1 | 0 |
| Rightmost eigenvalue | 1 | $\tau$ | 1 | $-\bar{\tau}$ |
| Largest eigenvalue | 0 | 1 | 1 | 0 |

and they satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq \left|\frac{\tilde{\theta}-\tau}{\tilde{\theta}+\bar{\tau}}\right| \|(A+\bar{\tau}B)\tilde{\mathbf{x}}\|. \tag{12}$$

However our implementation follows the next approach. The Galerkin conditions associated with the harmonic extraction and their variants can be generalized in the parametrized Galerkin condition [Hochstenbach 2005b, §5.1]

$$(\alpha A - \beta B)\tilde{\mathbf{x}} - \xi(\gamma A - \delta B)\tilde{\mathbf{x}} \perp \mathcal{W}' := (\alpha A - \beta B)\mathcal{V}, \quad \text{with} \quad \xi = \frac{\alpha\tilde{\theta}-\beta}{\gamma\tilde{\theta}-\delta}. \tag{13}$$

The values of $\alpha$, $\beta$, $\gamma$ and $\delta$ corresponding to the extraction methods are detailed in Table II. The projected eigenvalue problem associated with (13) is

$$W^*(\alpha A - \beta B)V\mathbf{u} = \xi W^*(\gamma A - \delta B)V\mathbf{u}, \quad \text{with} \quad (\alpha A - \beta B)V = WR, \tag{14}$$

where $W$ has orthonormal columns and $R$ is upper triangular. The error associated with the approximate eigenpairs is bounded by

$$\|(\alpha A - \beta B)\tilde{\mathbf{x}}\| \leq |\xi|\|(\gamma A + \delta B)\tilde{\mathbf{x}}\|. \tag{15}$$

This approach is concreted in Algorithm 3, by solving a problem equivalent to (14) at step 5, and computing $W$ as a basis of $\alpha A - \beta B$ at steps 2 and 21. The implementation does it by setting

— $W \leftarrow$ orthonormalize$((\alpha A - \beta B)V)$ in step 2, and
— $W^0 \leftarrow (\alpha A - \beta B)V_{m:m+s}$ in step 21.

### 2.3. Subspace Expansions

The first subspace expansion in the context of the Davidson methods was proposed with classical Davidson [Davidson 1975]. The method tries to compute the smallest eigenpairs of a standard Hermitian problem expanding the search subspace with $\mathbf{d}$ satisfying,

$$(\mathbf{diag}(A) - \tilde{\theta}I)\mathbf{d} = \mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}\tilde{\mathbf{x}}. \tag{16}$$

Subsequent developments [Morgan and Scott 1986; Natarajan and Vanderbilt 1989; Morgan 1990] tried to understand (16) as an iterative linear equation solver step of the system

$$(A - \tilde{\theta}I)\mathbf{d} = \mathbf{r}. \tag{17}$$

Then Generalized Davidson was proposed, a more general expansion also useful for non-Hermitian and generalized problems that introduced the use of fast, approximate inverses (the preconditioners, $K$),

$$\mathbf{d} = K^{-1}\mathbf{r}, \qquad K \approx A - \tilde{\theta}B. \tag{18}$$

The Olsen variant [Olsen et al. 1990] attempts to avoid the possible stagnation of the method when the resulting vector from the expansion $\mathbf{d}$ and the approximate eigenvector $\tilde{\mathbf{x}}$ are almost collinear (possibly because the preconditioner cannot improve the current approximation, see [Stathopoulos et al. 1995] for numerical experiments),

$$\mathbf{d} = -\left(I - \frac{K^{-1}B\tilde{\mathbf{x}}\tilde{\mathbf{x}}^*}{\tilde{\mathbf{x}}^*K^{-1}B\tilde{\mathbf{x}}}\right)K^{-1}\mathbf{r}. \tag{19}$$

In Jacobi-Davidson [Sleijpen and van der Vorst 1996; 2000] the search subspace is expanded by the approximate solution of the Jacobi orthogonal correction equation, that obtains a correction $\mathbf{d}$ orthogonal to the selected approximate eigenvector $\tilde{\mathbf{x}}$. In [Sleijpen et al. 1996] it is extended to generalized eigenproblems (and polynomial problems) and adapted to the use of a preconditioner (see [Sleijpen et al. 1996, Theorem 7.3]),

$$PK^{-1}(A-\theta B)P\mathbf{d} = -PK^{-1}\mathbf{r}, \quad \text{with } \mathbf{d}\perp\mathbf{z} \qquad \text{where } P = I - \frac{K^{-1}\mathbf{y}\mathbf{z}^*}{\mathbf{z}^*K^{-1}\mathbf{y}}. \tag{20}$$

Also, some theorems have been proposed about the convergence speed depending on the values of $\mathbf{z}$ and $\mathbf{y}$:

— For $A$ and $B$ Hermitian and $\lambda\in\mathbb{R}$, the choice of $\mathbf{z}=\tilde{\mathbf{x}}$ and any $\mathbf{y}$ such that $\mathbf{y}\not\perp\tilde{\mathbf{x}}$, implies quadratic convergence [Sleijpen et al. 1996, Remark 3.4].
— The choice of $\mathbf{y}=B\tilde{\mathbf{x}}$ and $\mathbf{z}$ such that $\mathbf{z}\not\perp\tilde{\mathbf{x}}$ leads to quadratic convergence [Sleijpen et al. 1996, Theorem 3.2], also $\mathbf{y}=\tilde{\theta}A\tilde{\mathbf{x}}+B\tilde{\mathbf{x}}$ leads to quadratic convergence [Sleijpen et al. 1996, Remark 3.1].
— The choice $\mathbf{y}=\mathbf{z}=\tilde{\mathbf{x}}$ implies superlinear convergence [Sleijpen et al. 1996, Theorem 3.4].

In SLEPc the correction equation is solved with $\mathbf{z}$ and $\mathbf{y}$ set to the approximate right eigenvector $\tilde{\mathbf{x}}$ and the corresponding vector from the test subspace, respectively. This solution can be found also in [Sleijpen et al. 1996; Fokkema et al. 1998].

### 2.4. Deflation

In works dealing with non-Hermitian problems and in early works of Jacobi-Davidson, the projector $P$ in the correction equation (20) is extended to deflate also against the converged pairs (see [Sleijpen and van der Vorst 1996; Fokkema et al. 1998])

$$P = I - K^{-1}\widehat{Y}(\widehat{Z}^*K^{-1}\widehat{Y})^{-1}\widehat{Z}^*, \quad \text{where } \widehat{Y}=[Y \quad \widetilde{Y}],\ \widehat{Z}=[X \quad \widetilde{X}]. \tag{21}$$

The extended projector avoids that the computed correction $\mathbf{d}$ converges toward the previously locked invariant subspace instead of new directions that could enrich the selected eigenvector. An extreme example is illustrated in [Fokkema et al. 1998, §4.5] in which the convergence is slowed down and finally stagnated due to the lack of newly produced directions.

The theoretical justification for this stagnation is presented next following [Fokkema et al. 1998, §3.4]. Consider the standard eigenvalue problem with matrix $A$ and eigenpairs $(\hat{\lambda}_i,\hat{\mathbf{x}}_i)$ and the approximate eigenpair $(\tilde{\theta},\tilde{\mathbf{x}})$ converging toward $(\hat{\lambda}_1,\hat{\mathbf{x}}_1)$. The exact solution of the correction equation (20) with $\mathbf{z}=\mathbf{y}=\tilde{\mathbf{x}}$ is given by [Sleijpen and van der Vorst 1996, §4.1]

$$\mathbf{d} = -\tilde{\mathbf{x}} + \epsilon(A-\tilde{\theta}I)^{-1}\tilde{\mathbf{x}}, \qquad \epsilon = (\tilde{\mathbf{x}}^*(A-\tilde{\theta}I)^{-1}\tilde{\mathbf{x}})^{-1}. \tag{22}$$

Considering the projection of $\tilde{\mathbf{x}}$ onto $\hat{\mathbf{x}}_i$ and the constraint $\mathbf{d} \perp \tilde{\mathbf{x}}$ results in

$$\mathbf{d} \approx \sum_{i \neq 1} \frac{\hat{\mathbf{x}}_i^* \tilde{\mathbf{x}}}{\hat{\lambda}_i - \tilde{\theta}} \hat{\mathbf{x}}_i. \tag{23}$$

Hence the directions $\hat{\mathbf{x}}_i$ of eigenvalues $\hat{\lambda}_i$ closest to $\tilde{\theta}$ become dominant. In that way, the convergence rate of $(\tilde{\theta}, \tilde{\mathbf{x}})$ increases as $\tilde{\theta}$ gets closer to $\hat{\lambda}_1$. However, suppose that a good approximation to $(\hat{\lambda}_1, \hat{\mathbf{x}}_1)$ was obtained and the next closest pair $(\hat{\lambda}_2, \hat{\mathbf{x}}_2)$ is sought. If the two pairs are approximately at the same *distance* to $(\tilde{\theta}, \tilde{\mathbf{x}})$, that is $|\hat{\lambda}_1 - \tilde{\theta}| \approx |\hat{\lambda}_2 - \tilde{\theta}|$ and $\hat{\mathbf{x}}_1^* \mathbf{x} \approx \hat{\mathbf{x}}_2^* \mathbf{x}$, or even worse, the first pair is the closest one, then we expect that $\mathbf{d}$ has many unwanted components of $\hat{\mathbf{x}}_1$. This can justify the implementation of the deflation in the correction equation.

On the other hand, in [Stathopoulos and McCombs 2007] the authors discuss the correction equation without the presence of the converged vectors in the context of a method for standard Hermitian problems (called JDQMR-000), showing some examples where it outperforms the deflation discussed above. Notice that the cost of applying the projector with converged vectors can become expensive when many pairs are sought: a cost of $\mathcal{O}(kn)$ per application for the simplest projectors and up to $\mathcal{O}(kn + k^3)$ for oblique projectors, when $k$ vectors are locked.

Our proposal is slightly more flexible and robust, allowing the user to limit the maximum number of converged vectors in the projector (a parameter called `pwindow` in §3.4). This approach can be useful in problems with presence of close eigenvalues, if a bound of the size of these clusters is available *a priori*.

### 2.5. Restarting

The maximum dimension of the subspace bases $V$ and $W$ is restricted with the parameter $m_{\max}$, limiting the cost of maintaining them orthogonal, that in general is one of the most expensive parts of the method, along with the matrix-vector product and the computation of the expansion. Too low values of $m_{\max}$ may prevent the convergence and too high values may affect the global performance negatively, and its optimal value depends on the application and the expansion employed.

From the methods that have been proposed to minimize the negative impact of restarting on convergence, we have incorporated in SLEPc (i) the thick restart technique [Stathopoulos et al. 1998], also called GD($m_{\min}$,$m_{\max}$), that restarts $V$ with a subspace basis that contains the best $m_{\min}$ current approximate eigenvectors; and (ii) its combination with a generalized CG-based restart [Stathopoulos and Saad 1998], resulting in GD($m_{\min}$,$m_{\max}$)+$k$, that enriches the thick restart basis with the best $k$ vectors from the previous iteration. For seeking extreme eigenpairs in standard Hermitian problems, the latter technique has theoretical [Stathopoulos and Saad 1998, §4] and practical [Stathopoulos and Saad 1998; Stathopoulos 2007] justifications.

### 2.6. Initializing the Search Subspace

An interesting advantage of the unconstrained Davidson's search subspace (in opposition to structured subspaces, like in the case of Krylov methods) is the possibility of starting with an available rough approximation of the sought eigenvectors, for instance in applications where the solution of a previous similar eigenproblem can be used or an analytic solution of a simpler problem is provided.

However, in practice initializing the search subspace with initial solutions $X_0$ is not enough to improve the convergence unless it is sufficiently close to the exact solutions. Alternatively, better results may be obtained by initializing the search subspace with

a Krylov subspace generated by the operator $K^{-1}(A-\tau B)$ and $X_0$. An example of use and performance is given in the applications of §5.3 and §5.4.

### 2.7. Stability

The common approach to the analysis of the quality of an obtained eigenpair $(\tilde{\theta},\tilde{\mathbf{x}})$ is in terms of the forward error (the difference between $(\tilde{\theta},\tilde{\mathbf{x}})$ and the exact solution $(\hat{\lambda}_1,\hat{\mathbf{x}}_1)$) approximated by the product of the backward error (the difference between the original problem $(A,B)$ and the *closest* problem $(\tilde{A},\tilde{B})$ that has $(\tilde{\theta},\tilde{\mathbf{x}})$ as exact solution, *i.e.*, $\tilde{A}\tilde{\mathbf{x}}=\tilde{\theta}\tilde{B}\tilde{\mathbf{x}}$) and the conditioning (how sensitive is $(\hat{\lambda}_1,\hat{\mathbf{x}}_1)$ to changes in the original problem $(A,B)$). It turns out that using a 2-norm metric the backward error of individual eigenvalues and eigenvectors can be bounded by their associated residual norms.

For Hermitian problems there are easy and practical expressions to bound the conditioning of eigenvalues and eigenvectors, and hence also for bounding the forward errors like the following ones for positive definite $B$ [Bai et al. 2000, §5.7.1],

$$|\tilde{\theta}-\hat{\lambda}_1| \leq \frac{\|\mathbf{r}\|^2_{B^{-1}}}{\delta\|\tilde{\mathbf{x}}\|^2_B} \leq \|B^{-1}\|_2^2\frac{\|\mathbf{r}\|_2^2}{\delta}, \tag{24}$$

$$\sin\angle(\hat{\mathbf{x}}_1,\tilde{\mathbf{x}}) \leq \|B^{-1}\|_2\sqrt{2\kappa(B)}\frac{\|\mathbf{r}\|_2}{\delta}, \tag{25}$$

where $\mathbf{r}=A\tilde{\mathbf{x}}-\tilde{\theta}B\tilde{\mathbf{x}}$ with $\|\tilde{\mathbf{x}}\|_2=1$, $\delta$ is the minimum distance between $\hat{\lambda}_1$ and any other eigenvalue $\hat{\lambda}_i$, and $\kappa(B)=\|B\|_2\|B^{-1}\|_2$ is the condition number of $B$. In Algorithm 2, $\|B^{-1}\|$ and $\kappa(B)$ can be estimated by making use of the largest and smallest $B$-inner product values from the $B$-orthogonalization, and $\delta$ can be approximated from the distance between $\tilde{\theta}$ and the rest of computed Ritz values. This leads to a cheap (although rough) approximation of the above error bounds.

In the presence of eigenvalue clusters (*i.e.*, several eigenpairs with eigenvalues close to each other, that in the extreme case may be the same), the eigenvalue accuracy is still bounded by its residual (there is a similar expression to (24) without the $\delta$ term), but the eigenvectors accuracy is not because in a cluster they are sensitive to perturbations. Nevertheless the eigenspace associated with the cluster (the subspace spanned by the eigenvectors pertaining to eigenvalues in the cluster) can be computed. In the above algorithms, this can be done by setting the block size $s$ to the maximum target cluster size and using a convergence criterion based on the residual norm associated with the eigenspace, $\|R_{1:s}\|_2=\|A\tilde{X}_{1:s}-B\tilde{X}_{1:s}\tilde{\Theta}_{1:s}\|_2$.

However for non-Hermitian problems an accurate bound of the error requires the full (generalized) Schur decomposition of the pair [Bai et al. 2000, §8.8], although the residual can still bound the backward error. In fact, the residual plays an important role in measuring the convergence of the iterative methods for solving eigenvalue problems, both theoretically (*e.g.*, [Morgan 1991; Fokkema et al. 1998; Hochstenbach and Notay 2006; Freitag and Spence 2007]) and practically (almost all solvers in software such as ARPACK, PRIMME, BLOPEX and Anasazi have a convergence criterion based on the residual).

We remark that in the case of degenerate eigenvalues (with algebraic multiplicity larger than one) our solvers do not have special difficulties. In the Hermitian case, if the block size is smaller than the actual multiplicity then a few multiple eigenvalues converge and the deflation mechanism allows the remaining copies to arise in the next restart. In non-Hermitian problems we have a similar behaviour, with the peculiarity that for defective eigenvalues at most a multiplicity equal to the geometric multiplicity will be obtained.

## 3. SLEPc EIGENSOLVER DESIGN

The two following subsections depict SLEPc, the library in which the previously outlined variants of the Davidson methods are implemented, and PETSc, the framework on which SLEPc relies. Then related software efforts for solving large-scale problems with Davidson solvers are presented. And the section ends with a detailed description of the user interface of the Davidson solvers in SLEPc.

### 3.1. PETSc Description

PETSc provides implementations of basic linear algebra operations with vectors and matrices, as well as tools for the solution of linear and non-linear systems of equations, all this enhanced with the support for distributed memory parallel computing platforms as well as an incipient support for emerging architectures such as shared memory clusters and GPUs. The interface is object-oriented, there are sets of functions in C (the class methods) accepting the same data type (representing a class), which encapsulates a pointer to a data structure of the class (that stores the object). The main advantage of this simple approach, over using a language with object-oriented support like C++, is the simplicity to invoke C functions from other languages. In fact PETSc can be used from C, C++, Fortran, Python and Matlab.

In general, the algorithm or the variant that will be used is determined by the *type*, that is set after the creation of the object. For instance, some of the available types for the KSP class, aimed at solving linear systems, are CG, GMRES and BiCGStab($\ell$).

The Vec and Mat classes gather different data structures for vectors and matrices. All vector objects are dense, but the Mat class offers a wide variety of matrix types like dense and sparse matrices, block diagonal matrices, FFT and user-defined implicit matrices, among others. Some types of vectors and matrices support high performance hardware, such as VecMPI, that distribute the vector and operations among MPI processes, and VecPThread and VecCUSP, that perform the operations by its distribution among POSIX threads and by launching GPU kernels, respectively. Types with similar capabilities can also be found in Mat.

Hence the parallelism and other high performance mechanisms are mostly transparent for other objects built on top of these, and also for the application programmer. The rest of classes such as linear, non-linear and time-stepping solvers, implement their methods without taking care of the underlying data-structures in matrices and vectors. The exception is the preconditioners class (PC), in which many of the types correspond to complete or partial factorizations (*e.g.*, LU, ILU, Cholesky and ICC) that are stored in ad-hoc Mat implementations.

For more information about the internal structure and the provided interfaces see [Balay et al. 1997; Balay et al. 2011].

### 3.2. Internal Structure of SLEPc

SLEPc provides a collection of state-of-the-art eigensolvers, SVD solvers and quadratic eigensolvers. Besides the Davidson-type solver addressed in this work, the class EPS provides an implementation of the Arnoldi, Lanczos and Krylov-Schur methods as well as wrappers to other libraries such as ARPACK, PRIMME and BLOPEX. Spectral transformations such as the shift-and-invert technique are encapsulated in the class ST, and are needed by Krylov methods to compute interior eigenvalues or to deal with matrix inversion in generalized eigenproblems. Class SVD provides algorithms for computing a partial singular value decomposition, for instance by solving an equivalent eigenvalue problem via EPS or by implementing native methods such as restarted Lanczos bidiagonalization. In the same way QEP objects can solve quadratic

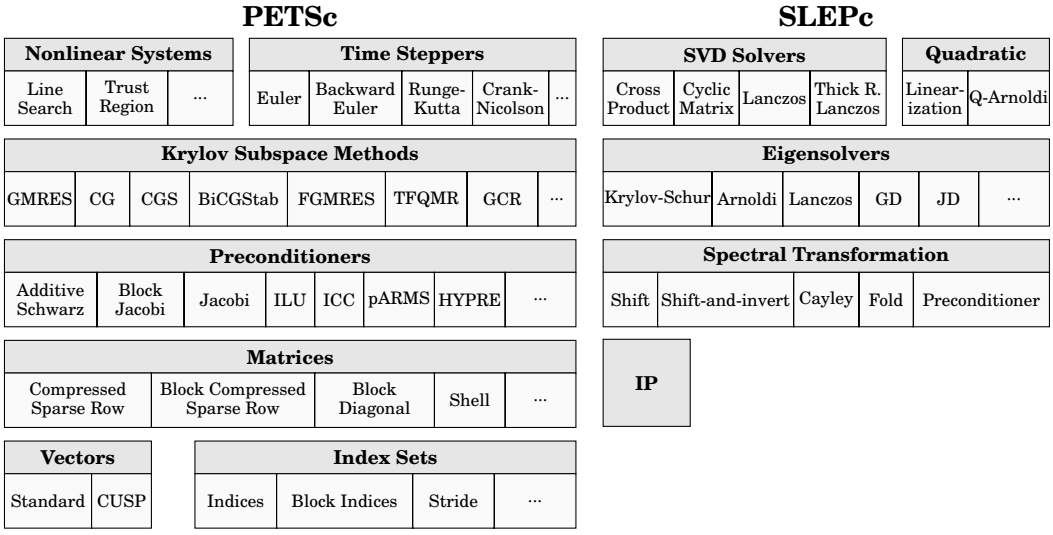


Fig. 1. Main classes in PETSc and SLEPc.

eigenproblems by using EPS to solve an eigenvalue problem resulting from linearization or directly with the Q-Arnoldi method.

Like other high-level classes, these SLEPc classes are built on the foundational classes Vec and Mat, and also KSP for the spectral transformations (matrix inversion is implemented via linear solves). As explained in §4.1, KSP will also be used in the solution of the correction equation in Jacobi-Davidson, and the preconditioner in the Generalized Davidson expansion. Orthogonalization and $B$-orthogonalization methods (like classical and modified Gram-Schmidt, optionally with selective reorthogonalization) are encapsulated in the class IP. For a complete description of SLEPc classes and usage, the reader is referred to [Campos et al. 2011].

PETSc and SLEPc delegate many dense operations to BLAS and LAPACK compatible libraries, such as the netlib reference implementation[2] or the ATLAS, MKL and ACML libraries. The delegated operations include frequent computations that account for most of the floating-point operations, like the addition and product of vectors during orthogonalization. Other delegated operations are performed on the projected problems, usually of rather small size, and they are important in terms of robustness of the algorithms, such as the computation of Schur decompositions in the extraction step of the algorithms in §2.

Furthermore SLEPc provides an interface for specifying operations with multivectors (a set of vectors that represent the columns of a thin tall matrix), giving the possibility to accelerate a sequence of level-2 BLAS operations by rewriting them as level-3 operations. The mechanism behind this consists in some functions to create PETSc Vec objects whose entries are stored contiguously in memory and to implement operations between such multivectors or multivectors with dense matrices. For instance, these functions are employed in the Davidson solvers in the creation of the projected matrices and the orthogonalization of the subspaces. As discussed below, interfaces to multivector operations are common in other libraries with Davidson methods.

[2] `http://netlib.org/blas/` and `http://netlib.org/lapack/`.

### 3.3. Comparison with the Design of other Parallel Davidson Software

Part of the design principles of Davidson methods in SLEPc can also be found in other libraries that implement related methods and share similar high performance aims, in particular Anasazi [Baker et al. 2009], BLOPEX [Knyazev et al. 2007] and PRIMME [Stathopoulos and McCombs 2010].

Anasazi is part of Trilinos, a parallel object-oriented software framework for large-scale multi-physics scientific applications. It was designed for being independent of the particular implementation of the underlying linear algebra primitives, in order to facilitate its incorporation into larger libraries and application codes. The Anasazi package contains a collection of eigensolvers that includes block Generalized Davidson and LOBPCG, among other solvers not related with Davidson. The eigensolvers provide an interface to establish the concrete implementations that will be used for vector and matrix operations, for the orthogonalization and for the stopping criterion.

BLOPEX is a stand-alone library that includes only a LOBPCG solver. The library is also distributed as part of the HYPRE[3] library for parallel preconditioning, and can optionally be used as a external solver in SLEPc. BLOPEX is designed with independence and interoperability goals similar to Anasazi, but without the possibility for customizing the orthogonalization method and the stopping criterion.

PRIMME implements a parametrized Davidson-type method, general enough to include algorithms ranging from Subspace Iteration to Jacobi-Davidson with several options about the correction equation. It uses its own distributed vectors with the solely support of BLAS and LAPACK, and a user-provided sum reduction operation. Currently, PRIMME only supports standard eigenproblems, although the interface is prepared for a matrix $B$. SLEPc also provides a wrapper to PRIMME as a external solver.

Table III summarizes the differences among these three libraries and our approach.

SLEPc's design goals lie between Anasazi/BLOPEX and PRIMME. On one hand, SLEPc eigensolvers use vectors, matrices and linear algebra primitives from PETSc. However, this PETSc-dependence does not reduce the software interoperability because PETSc classes allow for user-defined implementations. Anasazi and BLOPEX have an interface for multivector operations, and as mentioned before, SLEPc provides a simple interface for multivector operations such as inner-product $W^*V$ and the update $VU$, where $V$ and $W$ are multivectors and $U$ is a dense matrix.

The orthogonalization is also encapsulated in a SLEPc class and decoupled from the eigensolvers. Another simple mechanism to make the implementation independent of a given operation is to use callback functions. This is the way the convergence test and the sort routine can be replaced by user-defined code.

On the other hand, the code of Davidson-type methods in SLEPc is organized as a parametric multi-method, following PRIMME's approach. Hence, in practice, there is only one abstract object that contains all the Davidson code. By changing values in a C structure, the implementation behaves like either Algorithm 2 or 3. In the same way, the structure has variables to select the extraction and the expansion methods, to configure the restarting and to build the initial subspace.

### 3.4. Davidson-type Eigensolvers Interface

SLEPc offers two EPS solvers that implement Davidson-type methods, the Generalized Davidson (GD) and the Jacobi-Davidson (JD). As mentioned above, these objects are simply intermediate objects that access another abstract EPS object (abstract, in the sense that the end user cannot directly use it) that contains the implementation of the methods and techniques described in §2.

[3] `http://www.llnl.gov/casc/hypre`.

Table III. Summary of differences among BLOPEX, Anasazi, PRIMME, and SLEPc's Davidson, considering the implementation of the linear algebra operations (Vectors and Matrices), the possibility to change the orthogonalization routine (Orth.), the convergence test (Conv.) and the sort function (Sort), and how the Davidson solvers are organized.

| Framework | Vectors and Matrices | Customize | | | Eigensolvers |
|---|---|---|---|---|---|
| | | Orth. | Conv. | Sort | |
| BLOPEX | Abstract classes | - | - | - | Unique |
| Anasazi | | X | X | X | Separate |
| PRIMME | Hard code implementation | - | - | - | Parametrized Davidson Eigensolver |
| SLEPc | PETSc | X | X | X | |

The JD and GD solvers share the following interface with the rest of EPS objects:

— the problem matrices, with `EPSSetOperators`;
— the problem type like Hermitian, non-Hermitian, standard or generalized, with `EPSSetProblemType`, that selects between Algorithms 2 and 3;
— the subspace expansion method, determined by the object type with `EPSSetType`, that is (19) for GD and (20) for JD;
— the subspace extraction method, with `EPSSetExtraction`, that selects between the extraction techniques presented in §2.2;
— the sorting criterion for the eigenpairs computed by the extraction method, with `EPSSetWhichEigenpairs`, which can be relative to a target value, or with respect to the magnitude, the real or the imaginary part of the eigenvalue;
— the target value $\tau$ if interior eigenvalues are wanted, with `EPSSetTarget`, parameter that is also used in the extraction;
— the maximum size of the search and testing subspaces, $m_{\max}$ (see §2.5), and the number of wanted eigenpairs, $p$, with `EPSSetDimensions`, called `mpd` and `nev`, respectively;
— the convergence criterion, with `EPSSetConvergenceTest` (see §4.5);
— the tolerance for the convergence criterion and the maximum number of (outer) iterations, with `EPSSetTolerances` (see §4.5);
— the initial guess of the wanted invariant subspace, with `EPSSetInitialSpace` (see §2.6); and
— the subspace in which the eigenvectors are not wanted, with `EPSSetDeflationSpace` (see §2.4).

The specific options shared by the Davidson solvers are (the * has to be substituted by `GD` or `JD`, because there are different function names for GD and JD, for instance `EPS*SetBlockSize` is `EPSGDSetBlockSize` for GD and `EPSJDSetBlockSize` for JD):

— whether to enrich the initial subspace as explained in §2.6, with `EPS*SetKrylovStart`, activated by default;
— the block size $s$, with `EPS*SetBlockSize`, being 1 as default;
— the size of the subspace after restarting ($m_{\min}$) and the number of vectors saved from the previous iteration (in GD+$k$), with `EPS*SetRestart`, being the default values 6 and 0, respectively (see §2.5);
— the size of the initial search subspace, with `EPS*SetInitialSize`, being 5 as default (if the number of initial vectors provided by the user is smaller than this number, the initial subspace is filled with random vectors); and

— the maximum number of converged vectors to be included in the projectors (parameter `pwindow`, see §2.4), with `EPS*SetWindowSizes`, being 1 as default.

Moreover, the JD solver has two advanced options more that affect the convergence and performance of the solution of the correction equation, `EPSJDSetFix` (set to 0.01 by default) and `EPSJDSetConstantCorrectionTolerance` (deactivated by default), discussed in §4.1.

All preconditioned eigensolvers in SLEPc (JD and GD, but also the BLOPEX and PRIMME wrappers) are used in combination with the special ST object `Precond`. As other ST's (including shift-and-invert), it handles a linear solver (KSP) internally. The PC object in `Precond`'s KSP corresponds to the preconditioner used in the Generalized Davidson expansion or to the acceleration of the KSP that solves the Jacobi-Davidson correction equation. If the user does not provide a matrix as a basis for the preconditioner, with `STPrecondSetMatForPC`, the default preconditioner is built from $A - \tau B$ if interior eigenvalues are wanted, and $B$ if largest magnitude eigenvalues are wanted. In case the user does not select a preconditioning method Jacobi is used, if the diagonal is available. The method and the options for solving the correction equation are set in the KSP object. The default iterative solver for the JD correction equation is BiCGStab(2).

After the call to `EPSSolve`, the user can get individual converged eigenpairs with `EPSGetEigenpair`, or an orthogonal basis of the invariant subspace associated with them with `EPSGetInvariantSubspace`. Moreover, a basis of the test subspace associated with the converged pairs is accessible by the function `EPSGetInvariantSubspaceLeft`. Although not guaranteed, the test subspace may be a rough approximation to the left invariant subspace (if future versions of SLEPc include two-sided Jacobi-Davidson, more accurate approximations of the left invariant subspace will be obtained). Note that SLEPc does not currently make any provision to guarantee that no eigenvalue has been missed (except when computing all eigenvalues in an interval with Krylov methods). For Davidson methods, this could be achieved with iterative validation [McCombs and Stathopoulos 2006] in the case of Hermitian problems, but this is not implemented in SLEPc yet.

Figure 2 illustrates a simple example in C that computes the largest eigenvalue of a non-Hermitian matrix $A$, omitting the creation of the matrix, as well as the error checking and the functions to initialize and finalize the SLEPc environment. Notice that if PETSc is built in real arithmetic (that is, `PetscScalar` is not a complex type), SLEPc returns the real and imaginary part of eigenvalues (`kr` and `ki`) and eigenvectors (`xr` and `xi`) separately. The example uses GD, but JD could be selected simply by changing the call to `EPSSetType` in line 22.

Most of the above options are accessible via the command line. For instance, the code in Figure 2 can employ the GD solver to compute 10 eigenvalues with a relative tolerance of $10^{-8}$ by the execution of

```
$ ./exe -eps_type gd -eps_nev 10 -eps_tol 1e-8
```

or can employ JD for seeking the eigenvalues closest to 1 solving the correction equation with TFQMR (Transpose Free Quasi Minimal Residual) and accelerated with an ILU preconditioner:

```
$ ./exe -eps_type jd -eps_nev 10 -eps_target 1 -st_ksp_type tfqmr \
       -st_pc_type ilu
```

Another illustrative example is the parallel computation of the rightmost eigenvalues with 16 MPI processes, in which the rightmost harmonic extraction may accelerate the convergence:

```
#include "slepceps.h"
EPS         eps;       /*  eigensolver context  */
Mat         A;         /*  matrix of Ax=kx      */
Vec         xr, xi;    /*  eigenvector, x       */
PetscScalar kr, ki;    /*  eigenvalue, k        */
PetscInt    j, nconv;

/* Create the eigensolver context */
EPSCreate( PETSC_COMM_WORLD, &eps );
/* Set the problem matrices, B=I in this case */
EPSSetOperators( eps, A, PETSC_NULL );
/* Set the problem type: standard non-Hermitian */
EPSSetProblemType( eps, EPS_NHEP );
/* Specify the wanted eigenvalues */
EPSSetWhichEigenpairs( eps, EPS_LARGEST_MAGNITUDE );
/* Set number of eigenvalues to compute, 1, and default values for
   the maximum sizes of the search subspace and the projected problem */
EPSSetDimensions( eps, 1, PETSC_DECIDE, PETSC_DECIDE );
/* Set the convergence tolerance and the limit of iterations */
EPSSetTolerances( eps, 1e-10, 10000 );
/* Set the solver, Generalized Davidson */
EPSSetType( eps, EPSGD );
/* Apply other options indicated from the command line */
EPSSetFromOptions( eps );
/* Execute the solver */
EPSSolve( eps );
/* Get the number of converged pairs */
EPSGetConverged( eps, &nconv );
for (j=0;j<nconv;j++) {
  /* Get the j-th pair */
  EPSGetEigenpair( eps, j, &kr, &ki, xr, xi );
}
/* Destroy the eigensolver context */
EPSDestroy( eps );
```

Fig. 2. A short SLEPc example code that compute the eigenvalues of a non-Hermitian matrix.

```
$ mpirun -np 16 ./exe -eps_type gd -eps_nev 10 -eps_largest_real \
         -eps_harmonic_right -st_pc_type bjacobi
```

As we remark in §2, the optimal value of many parameters that have an important impact on the general performance depends on the application. As an example, the numbers referred to the search subspace like $m_{\min}$, $m_{\max}$ and its size after restarting can affect the global convergence and the performance of the orthogonalization step. Nevertheless most of the above options have default values that produce good performance in many cases, as the test battery results show (see §5.1). For instance, if the solver is configured to find eigenvalues close to a target $\tau$ (*e.g*, with the command line option `-eps_target`) in a non-Hermitian problem, the harmonic Rayleigh-Ritz extraction is activated and the pencil $A - \tau B$ is used for the preconditioner, of course if the user does not indicate other values. This feature facilitates the process of manually tuning a code to find optimal configurations.

## 4. IMPLEMENTATION DETAILS

In this section we include some discussions about the way we have implemented certain variants or aspects of the Davidson methods, such as the solution of the Jacobi-Davidson correction equation by an iterative Krylov solver (a KSP object), the treatment of complex numbers in real problems, how the data structures are distributed across processors and some considerations about the memory management.

### 4.1. Solution of the Correction Equation

Linear system (20) is solved when the Jacobi-Davidson expansion is selected. For that, a PETSc Krylov linear solver is employed (a KSP object), accelerated with preconditioners (PC object). Practical aspects of how to modify the correction equation and the projectors in order to use a preconditioner are summarized below (details can be found in [Sleijpen et al. 1998, §3.2]).

Most KSP solvers available in PETSc support left preconditioning (many of them exclusively), that is, they solve the linear system $M^{-1}\hat{A}\mathbf{x} = M^{-1}\hat{\mathbf{b}}$. In that case, (20) is equivalent to a linear system with coefficient matrix $\hat{A} = PK^{-1}(A-\theta B)$ and right hand side $\hat{\mathbf{b}} = -PK^{-1}\mathbf{r}$. If the initial guess is a zero vector, then the obtained approximate solution $\mathbf{d}$ will satisfy the constraint $\mathbf{d} \perp \mathbf{z}$.

The most frequently used KSP solver supporting right preconditioning ($\hat{A}M^{-1}M\mathbf{x} = \hat{\mathbf{b}}$) is FGMRES [Saad 1993], that is employed when the preconditioner can change during the iteration, for instance, when it performs a nested linear solve iteration as in pARMS [Li et al. 2003]. In the right preconditioning case, (20) is equivalent to a linear system with coefficient matrix $\hat{A} = Q(A - \tau B)$ where $Q$ can be $P$ or $I - \mathbf{z}\mathbf{z}^*$, the preconditioner $M^{-1} = PK^{-1}$ and right hand side $\hat{\mathbf{b}} = -\mathbf{r}$.

In both cases, the operator $\hat{A}$ and the preconditioner $M^{-1}$ are implemented implicitly (that is, only the matrix-vector product is defined) using a MatShell and a PCShell, respectively (these *shell* constructs are simply a way to encapsulate user-defined operations as a PETSc object).

In terms of computational cost, the weight of the solution of the correction equation can be heavy if too accurate solutions are requested. The trade-off between performance and global convergence is controlled in the stopping criterion, that the user can configure by setting the maximum number of iterations besides the relative and the absolute tolerances. In addition, unless `EPSJDSetConstantCorrectionTolerance` is invoked, the KSP stops when the residual of the linear system at iteration $j$, $\|\hat{\mathbf{r}}^{(j)}\|_2$, satisfies at outer iteration $i$ (counting from the beginning or from the last eigenpair converged)

$$\|\hat{\mathbf{r}}^{(j)}\|_2 \leq 2^{-i}\|\hat{\mathbf{r}}^{(0)}\|_2. \tag{26}$$

Notice that in PETSc the stopping criterion uses the preconditioned residual by default. This dynamic criterion comes from the Newton methods and the use in Jacobi-Davidson is suggested in [Fokkema et al. 1998], and tested in [Genseberger 2010; Romero and Roman 2011].

It is well-know that in the first iterations of the Davidson-type methods the extraction method usually produces poor eigenpair approximations and the target $\tau$ may be a relatively closer approximation to an exact eigenvalue [Morgan and Scott 1986]. Therefore, until the residual norm associated with the selected eigenpair reaches a threshold value, so called *fix* (that can be set by `EPSJDSetFix`), the correction equation is solved with $\theta = \tau$ [Fokkema et al. 1998].

### 4.2. Real Arithmetic

In real non-symmetric problems the eigenvalues and the corresponding eigenvectors may be complex, or even in the convergence of a real eigenpair the approximate eigenpairs may be complex. Considering that PETSc does not support operations mixing real and complex matrices or vectors, a simple workaround would be to perform all the computations in complex arithmetic. However, this is wasteful not only in terms of memory requirements but also in computational efficiency since the effective opera-

tion throughput may be reduced up to 50% for sufficiently large problems in which the bottleneck is the bandwidth between the main memory and the processor.

Another possibility to avoid complex arithmetic is the use of real Schur decompositions, where all matrices involved are real and $2 \times 2$ blocks on the diagonal of quasi-triangular matrices are used to represent complex conjugate eigenvalues. Besides requiring half the storage of a complex Schur decomposition, another advantage of the real Schur form is that complex conjugate eigenvalues always appear together.

Thus SLEPc's Jacobi-Davidson has an implementation based on RJDQZ [van Noorden and Rommes 2007], that adapts the extraction process and the correction equation (20) to work with the real Schur form. Considering the preconditioning, the resulting correction equation for the selected complex conjugate eigenvalues $\tilde{\theta}^r \pm \tilde{\theta}^i i$ and the associated Schur vectors $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ is

$$\begin{bmatrix} P^B & 0 \\ 0 & P^B \end{bmatrix} \begin{bmatrix} K^{-1}(A-\tilde{\theta}^r B) & \tilde{\theta}^i K^{-1}B \\ -\tilde{\theta}^i K^{-1}B & K^{-1}(A-\tilde{\theta}^r B) \end{bmatrix} \begin{bmatrix} P^B & 0 \\ 0 & P^B \end{bmatrix} \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} = - \begin{bmatrix} P^B K^{-1}\mathbf{r}_1 \\ P^B K^{-1}\mathbf{r}_2 \end{bmatrix}, \tag{27}$$

where $P^B$ is the block version of the projector $P$ in (20),

$$P^B = I - K^{-1}Y(Z^*K^{-1}Y)^{-1}Z^*, \tag{28}$$

and the residual is computed as

$$\begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} = \begin{bmatrix} A-\tilde{\theta}^r B & \tilde{\theta}^i B \\ -\tilde{\theta}^i B & A-\tilde{\theta}^r B \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1 \\ \tilde{\mathbf{x}}_2 \end{bmatrix}. \tag{29}$$

This version of the correction equation admits the same stopping criterion as (20), because if $\mathbf{d}_1$ and $\mathbf{d}_2$ are obtained from the approximate solution of the correction equation (27) with a linear system residual norm $\varepsilon$, then also $\mathbf{d} = \mathbf{d}_1 + \mathbf{d}_2 i$ satisfies the correction equation (20) with the same residual norm tolerance. Moreover we do not expect a significant difference in the convergence of the iterative resolution of both equations because the condition number corresponding to the coefficient matrices are the same [van Noorden and Rommes 2007, Proposition 1].

If Jacobi-Davidson is activated, the KSP in ST is responsible for solving the correction equation, that can be

— the linear system (20), if either the problem and the selected eigenpair are both real or the problem is complex, or
— the double-sized linear system (27), if the problem is real, but the selected eigenpair is complex.

In the case of real problems, the KSP object would have to solve linear systems of different sizes, which implies reallocating the memory every time the system size changes. This problem can be neglected if the block size is one and the eigenpairs last many iterations before their convergence, because in general we do not expect an excessive number of jumps of a single approximate eigenvalue from real to complex (and vice versa). For applications that require many eigenvalues and they converge quickly, we have designed a special vector type VecComp with *virtually* the length of the double-sized system, which is employed by the KSP object. VecComp vectors are formed by two sub-vectors whose length is equal to the problem size, and only one is used when the approximate eigenvalue is real. Working with these vectors also simplifies the implementation of the double-sized coefficient matrix-vector product (which is implemented using the MatShell PETSc class, because the double-sized matrix is never built explicitly), that can be arranged as a 2-by-2 block product. This product also includes the projection and the preconditioner application. For that, the PC as-

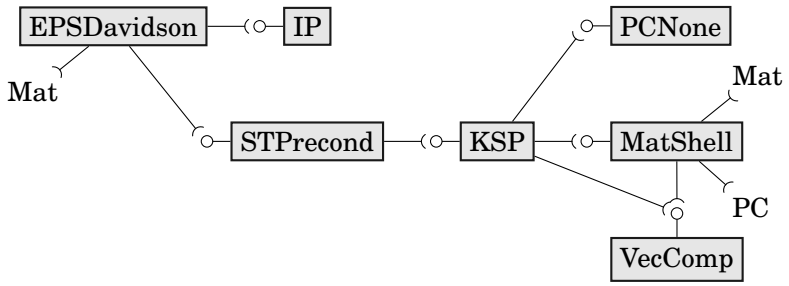


Fig. 3. Collaboration diagram for the SLEPc's Davidson solver when the Jacobi-Davidson expansion is selected.

sociated with the KSP object is detached and replaced by a dummy PC (PCNone). Figure 3 depicts the component diagram in this case.

### 4.3. Parallelization and Memory Management Details

The problem matrices $A$ and $B$ and the vectors of the same size, such as the search and test subspaces $V$ and $W$, and the converged invariant subspace $\widetilde{X}$, are distributed by blocks of rows among the processes, in the case of using MPI objects for vectors and matrices. The rest of the matrices and vectors with smaller size (bounded by $m_{\max}$) are implemented as sequential (non-MPI) objects, but are replicated in all nodes. For instance, this is the case of the projected matrices and the associated decompositions $\Theta$, $U$, $Z$, $S$ and $T$.

The operations involving distributed operands are parallelized, such as the $A$ and $B$ matrix-vector product, the subspaces updating, the computation of the projected problem matrices and the orthogonalization of the subspaces. In addition, these operations can also be accelerated by the GPU or parallelized for taking advantage of multi-cores, if the appropriate types are set for matrices and vectors. Currently these features are experimental in PETSc, but in the near future they are expected to be fully functional.

In terms of memory management, nearly all memory required by the eigensolver (the bases of the search and test subspaces, auxiliary vectors and work spaces) is allocated in a contiguous array, in order to reduce the memory management overhead. Each piece in which the allocated memory is divided is aligned properly.

PETSc does not support block application of matrices and preconditioners nor multivector operations, unlike other packages such as BLOPEX or Anasazi. However SLEPc offers some support for vector-vector operations in blocks, that the Davidson-type solvers employ except in the orthogonalization.

### 4.4. Subspace Orthogonalization

The orthogonalization of vectors can become an expensive step in variants with cheap expansion, *e.g.*, Generalized Davidson, so it is important to use a procedure that is efficient both sequentially and in parallel. The SLEPc class IP provides Classical and Modified Gram-Schmidt to carry out that process. The default configuration (used in §5) employs iterative CGS (which yields better parallel scaling and higher floating point operations throughput than MGS), with a DGKS-like re-orthogonalization criterion. See [Hernandez et al. 2007] for details.

This configuration is also used for maintaining $B$-orthogonal bases, although it is not clear that the re-orthogonalization criterion can be useful when using $B$-inner products with ill-conditioned $B$ [Kopal et al. 2012] (to avoid problems, it is possible in SLEPc to deactivate selective re-orthogonalization and use double orthogonalization instead). In any case, we present in Algorithm 4 a variant of iterative CGS that avoids the extra

**ALGORITHM 4:** Optimized Iterative Classical Gram-Schmidt with $B$-inner product.

**Input**: matrix $B$ of size $n$, $B$-orthonormal basis $V$, $V^B = BV$, vector $\mathbf{v}_0$
**Output**: $B$-orthonormal basis $[V\ \mathbf{v}]$ of $\mathrm{span}\{[V\ \mathbf{v}_0]\}$ and $\mathbf{v}^B = B\mathbf{v}$

```
v ← v_0 and v^B ← Bv_0
for i = 1, 2, 3 do
    h ← V^* v^B
    v ← v − Vh
    v^B ← v^B − V^B h
    Test criterion
end
v = v/‖v‖_B
```

matrix-vector products when the search subspace is $B$-orthogonalized, at the cost of additional storage for vectors $BV$. This variant is not activated by default, because it can only present an improvement when $B$ has bad matrix-vector performance.

### 4.5. Convergence Criterion for the Eigensolver

SLEPc monitors the residual norm associated with the approximate eigenpairs in order to detect the converged ones with respect to some criterion, *e.g.*, the default one is $\|r\|_2/|\tilde{\theta}| \leq \varepsilon$ for a given tolerance $\varepsilon$, or other criteria that can be defined by the user (in a function that the eigensolver will execute after every approximate pair is computed). When the problem is non-Hermitian, the solver works with approximate Schur vectors instead of eigenvectors, so the residual norms associated with the eigenpairs are not readily available. Our implementation first checks the convergence criterion with the residual associated with the Schur vector, and if it is passed, the criterion is checked again with the corresponding eigenvector residual. In this way, we avoid the costly computation of the eigenvector in each iteration unless it is close to convergence.

## 5. RESULTS

In this section, we present performance results of the SLEPc Davidson solvers. First sequential results in the context of a collection of small eigenproblems are shown comparing our implementations of GD and JD, as well as with other implementations such as Anasazi GD and PRIMME JD, besides with other methods such as LOBPCG in BLOPEX and inexact Krylov-Schur (where an iterative linear solver is used for the implicit inverse of the shift-and-invert operator). Then we show parallel results in the context of three large-scale scientific applications.

### 5.1. Test Battery

We compare the Davidson solvers in terms of execution time and number of preconditioner applications required by each one in the solution of a collection of problems.

The collection consists of standard and generalized problems whose matrices come from the University of Florida Sparse Matrix Collection [Davis and Hu 2011]. For each problem, the sequential performance of the solvers, in terms of number of matrix-vector products and time spent, is obtained computing 1 and 10 pairs with an absolute tolerance of $10^{-10}(\|A\|_2+|\tau|\|B\|_2)$ (with $\tau$ being the geometrical mean of the finite, non-zero eigenvalues of the problem) using either no preconditioner or one of the PETSc preconditioners Jacobi, ILU (or ICC for Hermitian problems), HYPRE and pARMS. The rest of the solver's parameters keep the default values: 5 random vectors as initial subspace, restart the search subspace with 6 vectors after the dimension reaches 17 or 26, respectively for seeking 1 or 10 pairs. The default linear solver for the JD correction

Table IV. Number of converged cases in each experiment.

| Solver | Total | None | Jacobi | ILU/ICC | HYPRE | pARMS |
|---|---|---|---|---|---|---|
| *Experiment I: generalized non-Hermitian, largest magnitude eigenvalues* | | | | | | |
| GD | 388 | 45 | 73 | 100 | 89 | 81 |
| JD BiCGStab(2) | 300 | 40 | 33 | 91 | 71 | 65 |
| Inexact Krylov-Schur | 31 | 0 | 0 | 12 | 1 | 18 |
| *Experiment II: standard and generalized non-Hermitian, eigenvalues closest to target* | | | | | | |
| GD | 599 | 108 | 112 | 211 | 120 | 165 |
| JD GMRES | 280 | 21 | 42 | 94 | 43 | 80 |
| JD BiCGStab(2) | 490 | 64 | 111 | 216 | 115 | 159 |
| Inexact Krylov-Schur | 248 | 0 | 30 | 87 | 53 | 78 |
| *Experiment III: generalized Hermitian, smallest real eigenvalues* | | | | | | |
| GD | 273 | 63 | 101 | 109 | | |
| GD Anasazi | 124 | 20 | 56 | 48 | | |
| JD BiCGStab(2) | 263 | 66 | 97 | 100 | | |
| BLOPEX | 228 | 32 | 104 | 92 | | |
| Inexact Krylov-Schur | 37 | 0 | 18 | 19 | | |
| *Experiment IV: standard Hermitian, eigenvalues closest to target* | | | | | | |
| GD | 873 | 339 | 298 | 236 | | |
| GD Anasazi | 726 | 363 | 357 | 6 | | |
| PRIMME JDQMR_Etol | 1005 | 405 | 219 | 309 | | |
| JD BiCGStab(2) | 768 | 224 | 310 | 234 | | |

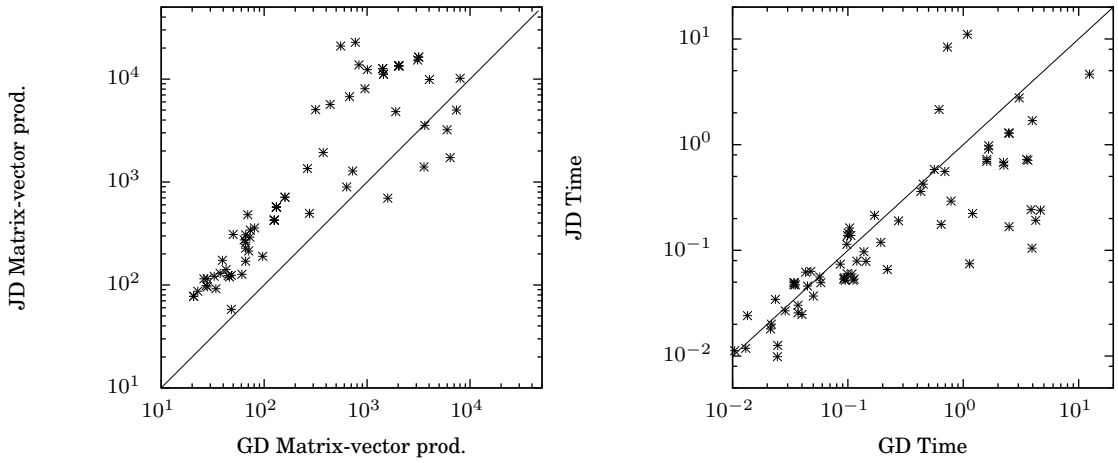


Fig. 4. Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc GD and JD computing the largest magnitude eigenvalues of generalized non-Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

equation is BiCGStab(2). Each of these configurations is executed twice with three different initial random vectors, discarding the slower one.

Table IV summarizes the number of cases in which the solvers obtained all the requested eigenpairs with less than 5050 and 10100 matrix-vector products for standard and generalized problems, respectively. We use matrix-vector products because it is unfair to compare GD and JD in terms of the number of outer iterations. Instead, it is more natural to roughly compare the number of iterations used by GD with the accumulated number of inner iterations employed by JD.

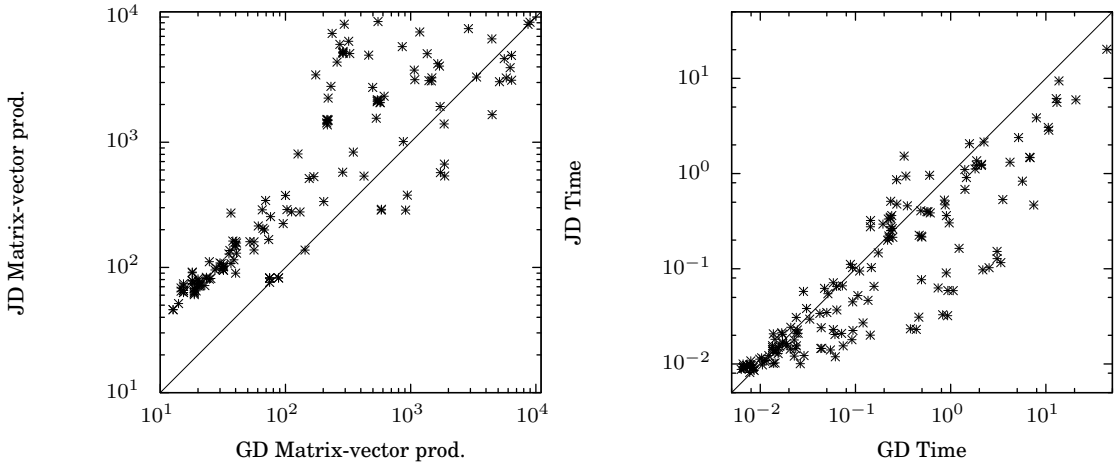


Fig. 5. Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc GD and JD computing the eigenvalues closest to a target in non-Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

Experiment I takes results of the SLEPc solvers GD and JD computing the largest magnitude eigenpairs of a group of 25 generalized non-Hermitian problems. JD solves slightly more problems than GD, and many of them faster (see Figure 4, right). However, in terms of matrix-vector products, JD generally requires more products than GD (see Figure 4, left).

Experiment II collects the performance of the solvers with a group of 52 standard and generalized non-Hermitian problems while computing the eigenvalues closest to a target in the interior of the spectrum. In absolute terms, GD successfully converges in more cases than JD (599 vs 490). In part, this is due to the few converged problems obtained by JD using sophisticated preconditioners (ILU, HYPRE and pARMS), compared with the JD results when no preconditioning is used, and with the GD results for those preconditioners. Apparently, the use of GMRES for solving the correction equation does not improve the results. Nevertheless, GD needs less matrix-vector products, but JD is generally faster, as in the previous experiment (see Figure 5).

In Experiment III, the eigenvalues with smallest real part are computed in 30 generalized Hermitian problems. We observe that both SLEPc solvers perform almost as well as LOBPCG in BLOPEX, a specific method for this case.

Experiment IV compares SLEPc GD and JD solvers with PRIMME's JDQMR_Etol, in which the JD correction equation is solved by QMR with an ad-hoc stopping criterion [Stathopoulos 2007, §3.3]. In this case, the eigenvalues closest to an interior target are computed in a group of 99 standard Hermitian problems. We observe the same trend concerning the number of converged pairs by GD and JD as in experiment II. PRIMME JD obtains better figures than SLEPc solvers, possibly due to the effectiveness of the PRIMME stopping criterion, that reduces significantly the number of matrix-vectors products (see Figure 6, left). However this PRIMME advantage is less important considering the total time spent by the solvers (see Figure 6, right).

In the experiment involving Hermitian problems we tested the GD implementation in Anasazi, configured similarly to the employed GD in SLEPc. We observed that the Anasazi solver obtains significantly worse performance when preconditioners are used,

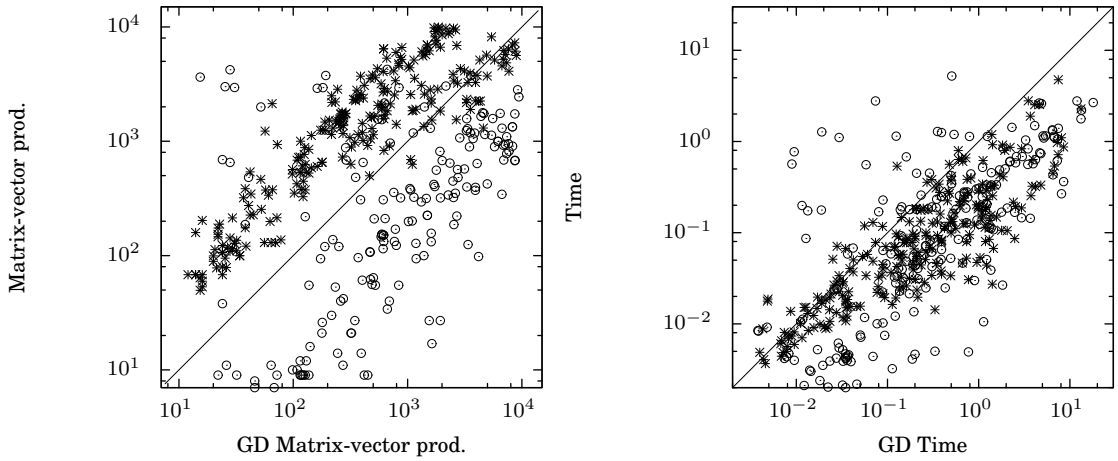


Fig. 6. Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc JD ($*$) and PRIMME JD ($\odot$), taking the results of SLEPc GD as a basis, when computing the eigenvalues closest to a target in standard Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

that may be due to the fact that Anasazi implements the basic Generalized Davidson (18), instead of the Olsen variant (19) (see §2.3), penalizing the convergence.

Finally, we also included a comparison with inexact Krylov-Schur available in SLEPc, which shows in general worse performance than the Davidson methods. However we used a simple stopping criterion in the resolution of the linear system associated with the shift-and-invert transformation (GMRES with default options except that relative tolerance is set to $10^{-12}$), and maybe better results could be obtained using a more sophisticated one.

### 5.2. Application 1: Incompressible Flow Analysis

The next problem comes from modelling a 2D incompressible fluid flow over a step. We use `navier_testproblem` from package IFISS (Incompressible Flow Iterative Solution Software, [Elman et al. 2007]) to generate the problem matrices corresponding to a Navier-Stokes problem of a fluid with viscosity $1/50$ flowing in a rectangular domain of $[-1,5]\times[-1,1]$ discretized with biquadratic/bilinear ($Q_2-Q_1$) finite elements with element width of $1/16$ (setting the grid parameter to 6). That results in a generalized non-Hermitian eigenvalue problem of size 13,682, with this structure

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \lambda \begin{bmatrix} B_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}.$$

Its resolution is numerically challenging because many eigenvalues are infinite (an 11% in this case), due to rank deficiency of $B$, and the eigenvalues nearest to zero are very close together (see Figure 7, left). The target eigenvalues are those closest to the imaginary axis. Since all the eigenvalues have positive real part, the approach employed was to configure the solver to obtain the eigenvalues closest to -5. JD and GD were tested to find 4 eigenvalues with an absolute tolerance of $10^{-8}|\tilde{\theta}|$ using the default options, except for $m_{\min}=30$, $m_{\max}=50$ and the pARMS preconditioner. In this case, GD computed the eigenpairs significantly faster than JD (GD required 2,720

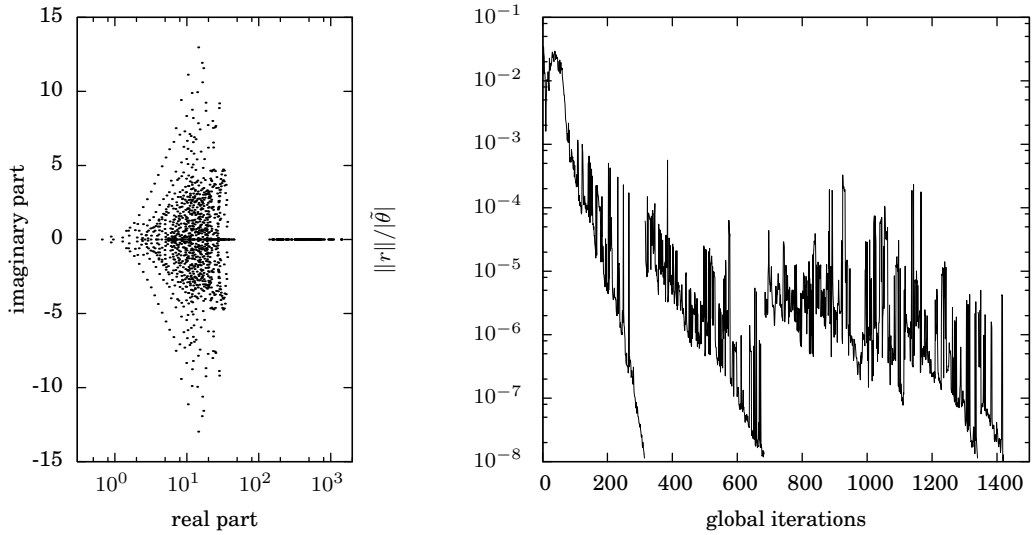


Fig. 7. Finite eigenvalues of a smaller problem (left) and convergence history employing the SLEPc GD solver (right) for the application described in §5.2.

matrix-vector products while JD needed as many as 664,144). Figure 7 (right) shows the convergence of GD.

### 5.3. Application 2: Unstable Modes of Turbulent Plasma

The plasma physics application GENE [Dannert and Jenko 2005] computes micro-instabilities in fusion plasma, solving the gyrokinetic equations, a set of nonlinear partial integro-differential equations in five-dimensional phase space by means of the method of lines. In certain analyses, a few rightmost eigenvalues of a large complex non-Hermitian linear operator (available through matrix-vector products) must be computed, which is computationally hard because eigenvalues with large imaginary part dominate the spectrum. The required rightmost eigenvectors correspond to the unstable modes of the linearized gyrokinetic equation,

$$\frac{\partial g}{\partial t} = \mathcal{L}[g], \tag{30}$$

that describes the time evolution of the distribution of the particles in the plasma.

A study of the sequential and the parallel performance of SLEPc's Jacobi-Davidson is presented in [Romero and Roman 2011], computing two rightmost eigenvalues using the harmonic extraction and solving the correction equation with BiCGStab(2) without preconditioner. The results reveal that the dynamic stopping criterion in the iterative solution of the JD correction equation (see §4.1) effectively improves the parallel performance (by reducing the overall number of reductions), and that the global performance has a strong dependence on the matrix-vector product performance.

An explicit sparse representation of the operator is not computationally feasible because of the high density of the resulting matrix. Instead, we opted for building a preconditioner based on a sparse part of the operator that still retains most of the information of the system. The results in [Merz et al. 2012] corresponding to this preconditioning illustrate an acceleration of more than 10 times faster using ASM (restricted Additive Schwarz Method) and more than 3 times using pARMS, both preconditioners scaling well (see Figure 8, left). Moreover the work presents a parameter scan test

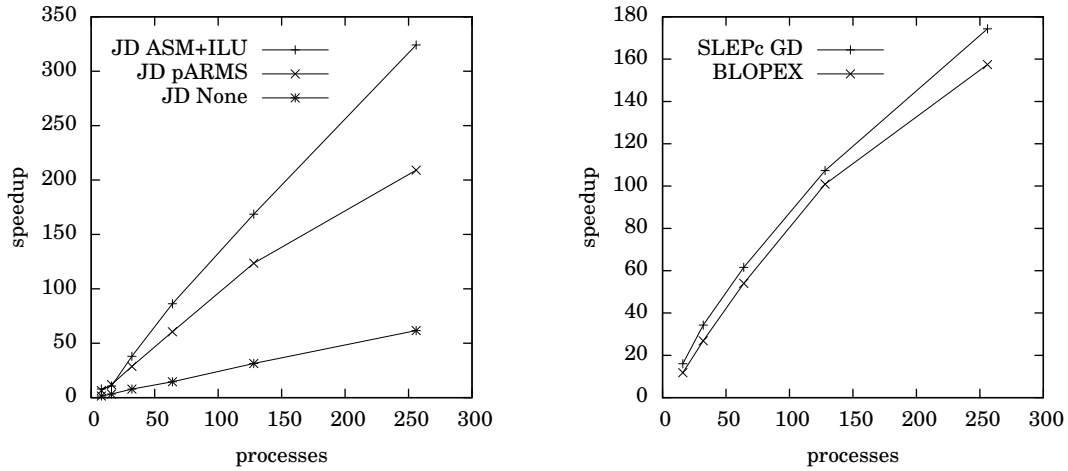


Fig. 8. Speedup examples corresponding to GENE (left) and the DFT code (right) employing the SLEPc JD and GD solvers, respectively. The right plot also shows results with the BLOPEX library. Speedups are computed relative to the fastest method.

case in which similar problems are solved, and the total time is reduced around 23% if the eigensolver's initial subspaces are taken from the previously obtained solutions enriched as described in §2.6.

### 5.4. Application 3: Electronic Configuration of Atoms

The Density Functional Theory (DFT) is one of the most popular methods that can be used to compute the electronic structure (principally the ground state) of atoms and molecules. The method requires the solution of the Schrödinger equation and the Poisson equation, that are coupled to each other. The computational approach consists in applying a self-consistent scheme, that is, the solution of the Schrödinger equation determines the next Poisson equation. The alternate solution of both equations is stopped when their results do not differ from the previous iterations under certain threshold.

After the discretization by the finite element method, the Poisson and Schrödinger equations result in a linear system and a generalized eigensystem, respectively, with large, sparse Hermitian matrices. Both problems are solved approximately, and in the case of the eigenvalue problem, the number of computed solutions depends on how many orbitals are sought.

The work [Young et al. 2013] presents a comparison of different refinement strategies for the meshes generated during the discretization, with a homemade code based on deal.II and SLEPc. The computation of the smallest eigenvalues (corresponding to the lowest energy orbitals) of the generalized Hermitian problems was done using GD with harmonic extraction and block Jacobi as the preconditioner for the expansion. Results for sequential and parallel performance are given in that paper, together with the acceleration produced by the subspace recycling within the self-consistent loop, resulting in a speedup of 4 with respect to no recycling.

With that code, we present a parallel performance comparison between GD (with Ritz extraction and optimized $B$-orthogonalization) and the LOBPCG method in BLOPEX, solving the electronic configuration of Beryllium, an atom with 4 protons

and 4 electrons. The number of computed eigenvalues is equal to the number of electrons. In this case, GD is slightly better than LOBPCG, as shown in Figure 8 (right).

## 6. CONCLUSIONS

This paper has introduced the implementation of Davidson methods in the context of the SLEPc library for the solution of Hermitian and non-Hermitian eigenvalue problems, both standard and generalized. The proposed solvers incorporate state-of-the-art expansion methods (such as Generalized Davidson and some variants of Jacobi-Davidson), extraction techniques (such as standard Rayleigh-Ritz and harmonic variants) and restart techniques (GD+$k$), exposing a number of parameters that allow for the adaptation of the solver to the characteristics of the problems. The solvers are robust and efficient by trying to maintain the structural properties of the original problem in the projected problem, and performing the operations in real arithmetic whenever possible.

The implementations are fully integrated in the PETSc framework, as the rest of SLEPc solvers, inheriting its benefits such as ease of solver customization via the command line (generally the optimal configuration is not straightforward), availability of a large variety of iterative linear solvers and preconditioners even from external packages, and high-performance computing capabilities, mostly MPI but also increasing support for novel architectures like multi-cores and GPUs.

In terms of practical use, the SLEPc Davidson solvers are competitive with respect to other free parallel libraries, and even provide new features like the support for non-Hermitian problems and harmonic extraction methods. We have addressed three relevant scientific computing applications, the analysis of incompressible fluid flow, the computation of unstable modes of plasma and electronic configurations of atoms, in which obtaining the corresponding eigenpairs is challenging for iterative solvers.

Near future works include the implementation of two-sided Jacobi-Davidson (for approximating the left eigenvectors besides the right ones at cubic convergence rate), and of robust, native Davidson methods for SVD and quadratic eigenproblems, although the current solvers can be used to solve those problems by means of solving the equivalent eigenproblem.

## REFERENCES

ARBENZ, P., BECKA, M., GEUS, R., HETMANIUK, U., AND MENGOTTI, T. 2006. On a parallel multilevel preconditioned Maxwell eigensolver. *Parallel Comput. 32,* 2, 157–165.

BAI, Z., DEMMEL, J., DONGARRA, J., RUHE, A., AND VAN DER VORST, H., Eds. 2000. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

BAKER, C. G., HETMANIUK, U. L., LEHOUCQ, R. B., AND THORNQUIST, H. K. 2009. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw. 36,* 3, 13:1–13:23.

BALAY, S., BROWN, J., BUSCHELMAN, K., EIJKHOUT, V., GROPP, W., KAUSHIK, D., KNEPLEY, M., MCINNES, L. C., SMITH, B., AND ZHANG, H. 2011. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.2, Argonne National Laboratory.

BALAY, S., GROPP, W. D., MCINNES, L. C., AND SMITH, B. F. 1997. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhaüser, 163–202.

BREBNER, M. A. AND GRAD, J. 1982. Eigenvalues of $Ax = \lambda Bx$ for real symmetric matrices $A$ and $B$ computed by reduction to a pseudosymmetric form and the HR process. *Linear Algebra Appl. 43*, 99–118.

CAMPOS, C., ROMAN, J. E., ROMERO, E., AND TOMAS, A. 2011. SLEPc users manual. Tech. Rep. DSIC-II/24/02 - Revision 3.2, D. Sistemes Informàtics i Computació, Universitat Politècnica de València. Available at http://www.grycap.upv.es/slepc.

DANNERT, T. AND JENKO, F. 2005. Gyrokinetic simulation of collisionless trapped-electron mode turbulence. *Phys. Plasmas 12,* 7, 072309.

DAVIDSON, E. R. 1975. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys. 17,* 1, 87–94.

DAVIS, T. A. AND HU, Y. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw. 38,* 1, 1:1–1:25.

ELMAN, H. C., RAMAGE, A., AND SILVESTER, D. J. 2007. Algorithm 866: IFISS, a Matlab toolbox for modelling incompressible flow. *ACM Trans. Math. Softw. 33,* 2. Article 14.

ERICSSON, T. AND RUHE, A. 1980. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp. 35,* 152, 1251–1268.

FERRONATO, M., JANNA, C., AND PINI, G. 2012. Efficient parallel solution to large-size sparse eigenproblems with block FSAI preconditioning. *Numer. Linear Algebra Appl. 19,* 5, 797–815.

FOKKEMA, D. R., SLEIJPEN, G. L. G., AND VAN DER VORST, H. A. 1998. Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput. 20,* 1, 94–125.

FREITAG, M. A. AND SPENCE, A. 2007. Convergence theory for inexact inverse iteration applied to the generalised nonsymmetric eigenproblem. *Electron. Trans. Numer. Anal. 28*, 40–64.

GENSEBERGER, M. 2010. Improving the parallel performance of a domain decomposition preconditioning technique in the Jacobi-Davidson method for large scale eigenvalue problems. *App. Numer. Math. 60,* 11, 1083–1099.

HERNANDEZ, V., ROMAN, J. E., AND TOMAS, A. 2007. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Comput. 33,* 7–8, 521–540.

HERNANDEZ, V., ROMAN, J. E., AND VIDAL, V. 2005. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Softw. 31,* 3, 351–362.

HEUVELINE, V., PHILIPPE, B., AND SADKANE, M. 1997. Parallel computation of spectral portrait of large matrices by Davidson type methods. *Numer. Algorithms 16,* 1, 55–75.

HOCHSTENBACH, M. E. 2005a. Generalizations of harmonic and refined Rayleigh-Ritz. *Electron. Trans. Numer. Anal. 20*, 235–252.

HOCHSTENBACH, M. E. 2005b. Variations on harmonic Rayleigh–Ritz for standard and generalized eigenproblems. Preprint, Department of Mathematics, Case Western Reserve University.

HOCHSTENBACH, M. E. AND NOTAY, Y. 2006. The Jacobi–Davidson method. *GAMM Mitt. 29,* 2, 368–382.

HWANG, F.-N., WEI, Z.-H., HUANG, T.-M., AND WANG, W. 2010. A parallel additive Schwarz preconditioned Jacobi-Davidson algorithm for polynomial eigenvalue problems in quantum dot simulation. *J. Comput. Phys. 229,* 8, 2932–2947.

KNYAZEV, A. V. 2001. Toward the optimal preconditioned eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient method. *SIAM J. Sci. Comput. 23,* 2, 517–541.

KNYAZEV, A. V., ARGENTATI, M. E., LASHUK, I., AND OVTCHINNIKOV, E. E. 2007. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in HYPRE and PETSc. *SIAM J. Sci. Comput. 29,* 5, 2224–2239.

KOPAL, J., ROZLOŽNÍK, M., TUMA, M., AND SMOKTUNOWICZ, A. 2012. Rounding error analysis of orthogonalization with a non-standard inner product. *Numer. Math. 52,* 4, 1035–1058.

KRESSNER, D. 2006. Block algorithms for reordering standard and generalized Schur forms. *ACM Trans. Math. Softw. 32,* 4, 521–532.

LEHOUCQ, R. B., SORENSEN, D. C., AND YANG, C. 1998. *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

LI, Z., SAAD, Y., AND SOSONKINA, M. 2003. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numer. Linear Algebra Appl. 10,* 5-6, 485–509.

MCCOMBS, J. R. AND STATHOPOULOS, A. 2006. Iterative validation of eigensolvers: a scheme for improving the reliability of Hermitian eigenvalue solvers. *SIAM J. Sci. Comput. 28,* 6, 2337–2358.

MERZ, F., KOWITZ, C., ROMERO, E., ROMAN, J. E., AND JENKO, F. 2012. Multi-dimensional gyrokinetic parameter studies based on eigenvalues computations. *Comput. Phys. Commun. 183,* 4, 922–930.

MORGAN, R. B. 1990. Davidson's method and preconditioning for generalized eigenvalue problems. *J. Comput. Phys. 89*, 241–245.

MORGAN, R. B. 1991. Computing interior eigenvalues of large matrices. *Linear Algebra Appl. 154–156*, 289–309.

MORGAN, R. B. AND SCOTT, D. S. 1986. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Statist. Comput. 7,* 3, 817–825.

NATARAJAN, R. AND VANDERBILT, D. 1989. A new iterative scheme for obtaining eigenvectors of large, real-symmetric matrices. *J. Comput. Phys. 82,* 1, 218–228.

NOOL, M. AND VAN DER PLOEG, A. 2000. A parallel Jacobi–Davidson-type method for solving large generalized eigenvalue problems in magnetohydrodynamics. *SIAM J. Sci. Comput. 22,* 1, 95–112.

OLSEN, J., JØRGENSEN, P., AND SIMONS, J. 1990. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chem. Phys. Lett. 169,* 6, 463–472.

PAIGE, C. C., PARLETT, B. N., AND VAN DER VORST, H. A. 1995. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numer. Linear Algebra Appl. 2,* 2, 115–133.

ROMERO, E. AND ROMAN, J. E. 2011. Computing subdominant unstable modes of turbulent plasma with a parallel Jacobi–Davidson eigensolver. *Concur. Comput.: Pract. Exp. 23,* 17, 2179–2191.

SAAD, Y. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput. 14,* 2, 461–469.

SLEIJPEN, G. L. G., BOOTEN, A. G. L., FOKKEMA, D. R., AND VAN DER VORST, H. A. 1996. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT 36,* 3, 595–633.

SLEIJPEN, G. L. G. AND VAN DER VORST, H. A. 1996. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl. 17,* 2, 401–425.

SLEIJPEN, G. L. G. AND VAN DER VORST, H. A. 2000. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM Rev. 42,* 2, 267–293.

SLEIJPEN, G. L. G., VAN DER VORST, H. A., AND MEIJERINK, E. 1998. Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal. 7*, 75–89.

STATHOPOULOS, A. 2007. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. *SIAM J. Sci. Comput. 29,* 2, 481–514.

STATHOPOULOS, A. AND MCCOMBS, J. R. 2007. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues. *SIAM J. Sci. Comput. 29,* 5, 2162–2188.

STATHOPOULOS, A. AND MCCOMBS, J. R. 2010. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Trans. Math. Softw. 37,* 2, 21:1–21:30.

STATHOPOULOS, A. AND SAAD, Y. 1998. Restarting techniques for the (Jacobi-)Davidson symmetric eigenvalue methods. *Electron. Trans. Numer. Anal. 7*, 163–181.

STATHOPOULOS, A., SAAD, Y., AND FISCHER, C. F. 1995. Robust preconditioning of large, sparse, symmetric eigenvalue problems. *J. Comput. Appl. Math. 64,* 3, 197–215.

STATHOPOULOS, A., SAAD, Y., AND WU, K. 1998. Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput. 19,* 1, 227–245.

STEWART, G. W. 2001. *Matrix Algorithms. Volume II: Eigensystems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

VAN DER VORST, H. A. 2002. Computational methods for large eigenvalue problems. In *Handbook of Numerical Analysis*, P. G. Ciarlet and J. L. Lions, Eds. Vol. VIII. Elsevier, Amsterdam, 3–179.

VAN DER VORST, H. A. 2004. Modern methods for the iterative computation of eigenpairs of matrices of high dimension. *Z. Angew. Math. Mech. 84,* 7, 444–451.

VAN NOORDEN, T. AND ROMMES, J. 2007. Computing a partial generalized real Schur form using the Jacobi–Davidson method. *Numer. Linear Algebra Appl. 14,* 3, 197–215.

YOUNG, T. D., ROMERO, E., AND ROMAN, J. E. 2013. Parallel finite element density functional computations exploiting grid refinement and subspace recycling. *Comput. Phys. Commun. 184,* 1, 66–72.

Received September 2011; revised ; accepted