SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems

VICENTE HERNANDEZ, JOSE E. ROMAN, and VICENTE VIDAL
Universidad Politécnica de Valencia

The Scalable Library for Eigenvalue Problem Computations (SLEPc) is a software library for computing a few eigenvalues and associated eigenvectors of a large sparse matrix or matrix pencil. It has been developed on top of PETSc and enforces the same programming paradigm.

The emphasis of the software is on methods and techniques appropriate for problems in which the associated matrices are sparse, for example, those arising after the discretization of partial differential equations. Therefore, most of the methods offered by the library are projection methods such as Arnoldi or Lanczos, or other methods with similar properties. SLEPc provides basic methods as well as more sophisticated algorithms. It also provides built-in support for spectral transformations such as the shift-and-invert technique. SLEPc is a general library in the sense that it covers standard and generalized eigenvalue problems, both Hermitian and non-Hermitian, with either real or complex arithmetic.

SLEPc can be easily applied to real world problems. To illustrate this, several case studies arising from real applications are presented and solved with SLEPc with little programming effort. The addressed problems include a matrix-free standard problem, a complex generalized problem, and a singular value decomposition. The implemented codes exhibit good properties regarding flexibility as well as parallel performance.

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

General Terms: Algorithms, Documentation

Additional Key Words and Phrases: Eigenvalue computation, spectral transform, singular values

1. INTRODUCTION

Eigenvalue problems are a very important class of linear algebra problems. The need for the numerical solution of these problems arises in many applications in science and engineering, for instance, in stability or vibrational analysis. In practical situations, these are usually formulated as large sparse eigenproblems.

This research was supported in part by the Valencian Science and Technology Agency with grant number CTIDB/2002/54.

Authors' addresses: Dept. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera, s/n, E-46022 Valencia, Spain; email: {vhernand,jroman,vvidal}@dsic.upv.es.

Computing eigenvalues is essentially more difficult than solving linear systems of equations. It has not been until recently that effective computational methods for eigenvalue problems have become available, together with some implementations of such methods. Available eigenvalue libraries are usually research codes focused on providing a robust implementation of a single method, and typically involve a number of parameters difficult to adjust by nonexperts. So it is quite common to solve the problem successfully only after several cycles of testing and parameter tuning. A desirable feature would be an easy way of experimenting with different parameter values as well as shifting from one method to another.

In the new era of scientific computing in which highly complex problems are addressed in large software projects developed by interdisciplinary teams, it is more and more important for software libraries to emphasize qualities such as interoperability and flexibility. Carefully designed libraries should be able to interoperate with other libraries, complementing each other. They should also allow the user to address problems formulated in a number of ways and easily adapt the solution strategy to the particular characteristics of the problem.

SLEPc is a library for eigenvalue problems that aims to achieve these goals by making use of the good properties of PETSc, such as extensibility, portability, scalability, flexibility, and interoperability (PETSc is described in Section 3). It offers a growing number of solution methods as well as interfaces to well-established eigenvalue packages. The text is organized as follows. Section 2 gives a short overview of the mathematical problem and the methods used to solve it. Section 3 reviews available software. Section 4 provides a brief description of SLEPc, together with some examples of usage. Section 5 presents three case studies illustrating the use of SLEPc in real applications. Finally, the last section gives some conclusions.

2. THE EIGENVALUE PROBLEM

In the standard formulation, the eigenvalue problem consists in the determination of $\lambda \in \mathcal{C}$ for which the equation

$$Ax = \lambda x \tag{1}$$

has a nontrivial solution, where $A \in \mathcal{C}^{n\times n}$ and $x \in \mathcal{C}^n$. The scalar λ and the vector x are called the *eigenvalue* and *eigenvector*, respectively. Quite often, the problem appears in generalized form, $Ax = \lambda Bx$. Other related linear algebra problems such as the quadratic eigenvalue problem or the singular value decomposition can be formulated as standard or generalized eigenproblems.

Many methods have been proposed to compute eigenvalues and eigenvectors. Some of them, such as the QR iteration, are not appropriate for large sparse matrices because they are based on modifying the matrix by certain transformations that destroy sparsity. On the other hand, most applications require only a few selected eigenvalues and not the entire spectrum. In this work, we consider only methods that compute a few eigenpairs of large sparse problems.

Methods for sparse eigenproblems [Bai et al. 2000] usually obtain the solution from the information generated by the application of the matrix to various vectors. Matrices are only involved in matrix-vector products. This preserves

sparsity and also allows the solution of problems in which matrices are not available explicitly.

The most basic method of this kind is the Power Iteration, in which an initial vector is repeatedly premultiplied by matrix A and conveniently normalized. Under certain conditions, this iteration converges to the eigenvector associated with the largest eigenvalue in module. After this eigenvector has converged, deflation techniques can be applied in order to retrieve the next one.

The Subspace Iteration is a generalization of the Power Method, in which the matrix is applied to a set of m vectors simultaneously, and orthogonality is enforced explicitly in order to avoid the convergence of all of them toward the same eigenvector. This method is often combined with a projection technique to compute approximations to the eigenpairs of matrix A, extracting them from a given low-dimensional subspace on which the problem is projected. In the case of the Subspace Iteration, this subspace is the one spanned by the set of iteration vectors.

The projection scheme is common to many other methods. In particular, so-called *Krylov methods* use a projection onto a Krylov subspace,

$$\mathcal{K}_m(A, v) \equiv \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\} \quad . \tag{2}$$

The most basic algorithms of this kind are the Arnoldi, Lanczos, and nonsymmetric Lanczos. These methods are the basis of a large family of algorithms.

The Arnoldi algorithm can be used for nonsymmetric problems. It computes approximations of invariant subspaces from Krylov subspaces of increasing size. Since the computational (and storage) cost grows with the size of these subspaces, the algorithm is typically restarted when a maximum is reached. Several restart alternatives have been proposed.

There are several issues that are common to the methods mentioned above. One of them is the orthogonalization strategy when constructing the basis. Several schemes are available with different behavior with respect to roundoff errors. Another important aspect is the management of convergence. Locking already converged eigenvalues can considerably reduce the cost of an algorithm.

Convergence problems can arise in the presence of clustered eigenvalues. Selecting a sufficiently large number of basis vectors can usually avoid the problem. However, convergence can still be very slow and acceleration techniques must be used. Usually, these techniques consist in computing eigenpairs of a transformed operator and then recovering the solution of the original problem.

The most commonly used spectral transformation is called *shift-and-invert* and operates with the matrix $(A - \sigma I)^{-1}$. The value of the shift, σ, is chosen so that the eigenvalues of interest are well separated in the transformed spectrum, thus leading to fast convergence. When using this approach, a linear system of equations must be solved whenever a matrix-vector product is required in the algorithm, and typically this computation must be quite accurate, often requiring the use of direct methods. Linear systems also appear in recently proposed eigensolvers such as Jacobi-Davidson, but in this context they need to be solved only approximately.

3. AVAILABLE SOFTWARE

There are several parallel software libraries which approach the problem by some variant of the methods mentioned above. The most complete of these libraries is ARPACK [Lehoucq et al. 1998], which implements the Arnoldi/Lanczos process with Implicit Restart for standard and generalized problems, in both real and complex arithmetic. Other available libraries are BLZPACK [Marques 1995], PLANSO [Wu and Simon 1997], and TRLAN [Wu and Simon 2001], which implement different flavors of the Lanczos method. All of them either require the user to provide a matrix-vector subroutine or to use a reverse communication scheme. These two interfacing strategies are format-independent and allow the solution of problems with implicit matrices. In addition to these, more recent algorithmic developments such as the Jacobi-Davidson method are also available in the form of Matlab implementations (some of them can be obtained via `http://www.cs.ucdavis.edu/~bai/ET/contents.html`).

Software design is becoming an increasingly important issue in scientific computing, not only to enable portability and reusability but also to pursue desirable features such as flexibility, extensibility, and interoperability. The objective is to be able to build codes for the solution of complex problems by putting together the functionality of different specialized software packages. This way makes life easier to the user, who can then focus more on research rather than wasting effort in the implementation of programs. Object-oriented design can provide important advantages over traditional design, for instance, hiding details of parallel execution, and allows the user to work at a higher level of abstraction. This is the approach used by PETSc [Balay et al. 2004], a toolkit for the solution of partial differential equations. PETSc is built around a variety of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets (`IS`), vectors (`Vec`), and matrices (`Mat`). Built on top of this foundation are various classes of solver objects, including linear, nonlinear, and time-stepping solvers. These solver objects encapsulate virtually all information regarding the solution procedure for a particular class of problems, including the local state and various options such as convergence tolerances, etc. Options can be specified by means of calls to subroutines in the source code and also as command-line arguments.

4. THE SLEPc LIBRARY

SLEPc, the Scalable Library for Eigenvalue Problem Computations, extends the functionality of PETSc to solve large sparse eigenvalue problems on parallel computers. It can be used for the solution of problems formulated in either standard or generalized form, as well as other related problems such as the singular value decomposition. It supports either real or complex arithmetic and includes solvers for both Hermitian and non-Hermitian problems.

The new functionality provided by the SLEPc library is organized around two objects, `EPS` and `ST`, as described next (see Hernandez et al. [2004] for details).

Table I. Operators Used in Each Spectral Transformation Mode

ST	Standard problem	Generalized problem
shift	$A+\sigma I$	$B^{-1}A+\sigma I$
sinvert	$(A-\sigma I)^{-1}$	$(A-\sigma B)^{-1}B$
cayley	$(A-\sigma I)^{-1}(A+\tau I)$	$(A-\sigma B)^{-1}(A+\tau B)$

4.1 EPS: Eigenvalue Problem Solver

The Eigenvalue Problem Solver (EPS) is the main object provided by SLEPc. It is used to specify an eigenvalue problem, either in standard or generalized form, and provides uniform and efficient access to all of the package's eigensolvers.

Apart from the usual object management functions (EPSCreate, EPSDestroy, EPSView, EPSSetFromOptions), the EPS object provides functions for setting several parameters such as the number of eigenvalues to compute, the dimension of the subspace, the requested tolerance, and the maximum number of iterations allowed. The user can also specify other things such as the orthogonalization technique or the portion of the spectrum of interest.

The solution of the problem is obtained in several steps. First, the matrices associated to the eigenproblem are specified via EPSSetOperators. Then, a call to EPSSolve is done which invokes the subroutine for the selected eigensolver. EPSGetConverged can be used afterward to determine how many of the requested eigenpairs have converged to working precision. Finally, EPSGetEigenpair is used to retrieve the eigenvalues and eigenvectors.

The solution method can be specified procedurally or via the command line. At the time of this writing, methods available in SLEPc include the Power Iteration with deflation, the Inverse Iteration, Rayleigh Quotient Iteration, Subspace Iteration with non-Hermitian projection and locking, and Arnoldi with explicit restart and deflation. More advanced methods are under development. In addition to these methods, there are also wrappers for ARPACK, BLZPACK, PLANSO, and TRLAN, and also a wrapper for some dense methods in LAPACK.

4.2 ST: Spectral Transformation

The other main SLEPc object is the Spectral Transformation (ST), which encapsulates the functionality required for acceleration techniques based on the transformation of the spectrum. The user does not usually need to create an ST object explicitly. Instead, every EPS object internally sets up an associated ST.

One of the design cornerstones of SLEPc is to separate spectral transformations from solution methods so that the user can specify any combination of them. To achieve this, all the eigensolvers contained in EPS must be implemented in a way that they are independent of which transformation has been selected by the user. That is, the solver algorithm has to work with a generic operator, whose actual form depends on the transformation used. After convergence, eigenvalues are transformed back appropriately, if necessary. Table I lists the operators used in each case, either for standard or generalized eigenproblems. By default, no trasformation is done or, equivalently, a shift of origin (shift) is applied with $\sigma = 0$. The other two options represent the shift-and-invert

(`sinvert`) and Cayley (`cayley`) transformations. In all cases, the value of the shift can be specified at run time.

The expressions shown in Table I are not built explicitly. Instead, the appropriate operations are carried out when applying the operator to a certain vector. The inverses imply the solution of a linear system of equations, which is managed by setting up an associated `KSP` object (PETSc's linear system solver). The user can control the behavior of this object by adjusting the appropriate options, as will be illustrated in the examples.

4.3 Supported Matrix Objects and Extensibility

Methods implemented in SLEPc merely require vector operations and matrix-vector products. In PETSc, mathematical objects such as vectors and matrices are defined in a uniform fashion without making unnecessary assumptions about internal representation. This implies that SLEPc can be used with any of the matrix and vector representations provided by PETSc, including dense and the different parallel sparse formats. Shell matrices are also supported, as illustrated in Section 5. These are matrices that do not require explicit storage of the component values. Instead, the user must provide subroutines for all the necessary matrix operations, typically only the application of the linear operator to a vector. Shell matrices are a simple mechanism of extensibility, in the sense that the package is extended with new user-defined matrix objects. Once the new matrix has been defined, it can be used by SLEPc like any other matrix.

SLEPc further supports extensibility by allowing application programmers to code their own subroutines for unimplemented features such as a new eigensolver. It is possible to register these new methods to the system and use them as the rest of standard subroutines. In the case of spectral transformations, `ST`, there is also a `shell` transformation provided for implementing user-defined transformations in a similar way as shell matrices.

4.4 Examples of Usage

The following C source code implements the solution of a simple standard eigenvalue problem by means of SLEPc. Code for creation of matrix A and error checking is omitted.

```
 1  #include "slepceps.h"
 2  EPS         eps;       /*  eigensolver context  */
 3  Mat         A;         /*  matrix of Ax=kx      */
 4  Vec         xr, xi;    /*  eigenvector, x       */
 5  PetscScalar kr, ki;    /*  eigenvalue, k        */
 6  int         j, nconv;
 7
 8  EPSCreate( PETSC_COMM_WORLD, &eps );
 9  EPSSetOperators( eps, A, PETSC_NULL );
10  EPSSetProblemType( eps, EPS_NHEP );
11  EPSSetFromOptions( eps );
12  EPSSolve( eps );
13  EPSGetConverged( eps, &nconv );
14  for (j=0;j<nconv;j++) { EPSGetEigenpair( eps, j, &kr, &ki, xr, xi ); }
15  EPSDestroy( eps );
```

All the operations of the program are done over a single EPS object, the eigenvalue problem solver, created in line 8. In line 9, the operator of the eigenproblem is specified to be the object A of type Mat, which could be a user-defined shell matrix. Then the problem type is set to be standard non-Hermitian. At this point, the value of the different options could be set by means of a function call such as EPSSetTolerances. After this, a call to EPSSetFromOptions should be made as in line 11. The effect of this call is that options specified at runtime in the command line are passed to the EPS object appropriately. In this way, the user can easily experiment with different combinations of options without having to recompile. Line 12 launches the solution algorithm. The subroutine that is actually invoked depends on which solver has been selected by the user. At the end, all the data associated to the solution of the eigenproblem is kept internally. Line 13 queries how many eigenpairs have converged to working precision. The solution of the eigenproblem is retrieved in line 14. Finally, the EPS context is destroyed.

The command line (MPI directives are omitted)

```
$ program -eps_nev 10 -eps_ncv 24
```

could be used to run the program, specifying the number of eigenvalues and the dimension of the subspace. In this other example, the solution method is given explicitly and the matrix is shifted with $\sigma = 0.5$

```
$ program -eps_type subspace -st_type shift -st_shift 0.5
          -eps_orthog_type mgs -eps_monitor
```

The last two keys select the modified Gram-Schmidt orthogonalization algorithm and instruct to activate the convergence monitor, respectively.

In the case of generalized problems, the user can additionally specify options relative to the solution of the linear systems. For example,

```
$ program -st_ksp_type gmres -st_pc_type ilu
```

In the above example, the prefix st_ is used to indicate an option for the linear system of equations associated to the ST object. In particular, the system is solved with GMRES preconditioned by ILU. Similarly, for using shift-and-invert:

```
$ program -st_type sinvert -st_shift 10 -st_pc_type jacobi
          -st_ksp_type cg -st_ksp_rtol 1e-7
```

In this last example, the value of the shift is $\sigma = 10$ and linear systems are solved via Conjugate Gradients with Jacobi preconditioning up to a precision of 10^{-7}. A detailed description of these options and others is included in the user's guide.

5. CASE STUDIES

In this section, three problems taken from real applications are described, and a sketch of a possible solution implemented with SLEPc is outlined. The objective is to illustrate how SLEPc can be successfully applied in different situations with little programming effort.

5.1 Lambda Modes of a Nuclear Reactor

The lambda modes analysis is a powerful tool for the safety analysis of nuclear reactors. It can be used to study the steady-state neutron flux distribution inside the reactor core (to detect subcritical modes responsible for regional instabilities) as well as for transient analysis. The latter case requires the computation of the lambda modes at each time step, thus making appropriate the use of a fast parallel eigensolver.

The lambda modes equation is a differential eigenvalue problem derived from the neutron diffusion equation. The discretization of the problem (see Hernandez et al. [2003] for details) leads to the following algebraic eigensystem:

$$L\psi = \frac{1}{\lambda} M\psi, \tag{3}$$

where ψ is related to the neutron flux distribution.

If neutron energy is discretized with a two-group approach, then matrices L and M have the following block structure:

$$\begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix}, \tag{4}$$

where L_{11} and L_{22} are nonsingular sparse matrices, and M_{11}, M_{12}, and L_{21} are diagonal matrices. L_{12} is zero because upscattering effects have not been considered. This allows rewriting the problem in a number of ways, such as

$$A\psi_1 = \lambda\psi_1, \tag{5}$$

where matrix A is given by

$$A = L_{11}^{-1}\left(M_{11} + M_{12}L_{22}^{-1}L_{21}\right). \tag{6}$$

This problem can be solved directly with the sample code given in Section 4.4, provided that an appropriate representation of matrix A in (6) is handled implicitly. This can be done with shell matrices in PETSc. Such a matrix can be created with

```
MatCreateShell( comm, n, n, N, N, (void*)ctx, A );
MatShellSetOperation( *A, MATOP_MULT, (void(*)())MatLambda_Mult );
```

where the matrix context `ctx` is a placeholder for all the relevant information:

```
1  typedef struct {
2  KSP        L11, L22;
3  Vec        w, L21, M11, M12;
4  } CTX_LAMBDA;
```

Here, L_{11} and L_{22} are represented as linear systems since they appear inverted in (6), and the diagonals of the other blocks are stored as vectors. In order to use this matrix with the EPS object, at least one operation must be defined, the matrix-vector multiplication, which can be accomplished with the following function:

```
5  int MatLambda_Mult( Mat A, Vec x, Vec y )
6  {
7  CTX_LAMBDA  *ctx;
```

```
 8  PetscScalar done = 1.0;
 9  MatShellGetContext( A, (void**)&ctx );
10  VecPointwiseMult( ctx->L21, x, ctx->w );        /*  w = L21 x     */
11  KSPSolve( ctx->L22, ctx->w, y );                /*  y = L22^-1 w  */
12  VecPointwiseMult( ctx->M12, y, ctx->w );        /*  w = M12 y     */
13  VecPointwiseMult( ctx->M11, x, y );             /*  y = M11 x     */
14  VecAXPY( &done, y, ctx->w );                    /*  w = w + y     */
15  KSPSolve( ctx->L11, ctx->w, y );                /*  y = L11^-1 w  */
16  }
```

5.2 Electromagnetic Cavity Resonators

The analysis of steady-state electromagnetic waves in cavities has many applications such as the design of microwave ovens, particle accelerators, or semianechoic chambers for electromagnetic compatibility studies. The mathematical model in this case is the eigenvalue problem solving the Maxwell equations in a bounded volume. In particular, we are addressing the source-free wave equation

$$\nabla \times (\hat{\nu}_r^{-1} \nabla \times \vec{\Psi}) - k_0^2 \hat{\sigma}_r \vec{\Psi} = 0, \tag{7}$$

where $\vec{\Psi}$ represents the electric field (or, alternatively, the magnetic field), and the permittivity and permeability tensors are denoted by $\hat{\sigma}_r$ and $\hat{\nu}_r$ (or vice versa).

The equation is discretized by means of the finite element method, which provides enough flexibility to handle strong variations in the solution as in the case of inhomogeneous cavities with several materials. Edge elements are used in order to avoid so-called *spurious modes*. Assembly of all the element contributions yields the following eigensystem:

$$R\dot{\Psi} - k_0^2 S\dot{\Psi} = 0, \tag{8}$$

where $\dot{\Psi}$ contains the edge values of the field. We are interested in the analysis of linear, anisotropic, and inhomogeneous cavities, possibly filled with lossy (absorbing) materials. Therefore, R and S can be complex matrices.

In order to solve the problem with the example code, the two matrices have to be passed to the EPS object, indicating that it is a generalized problem:

```
10 EPSSetOperators( eps, R, S );
```

In this way, the code is ready for computing the extreme eigenvalues or the internal ones if using shift-and-invert. However, in this particular application we are interested in computing the fundamental eigenfrequency (and a few of the following ones) together with the corresponding eigenfields. The fundamental eigenfrequency is the smallest nonzero eigenvalue, since matrix R has a high-dimensional null space. Therefore, the zero eigenvalues must be avoided during the computation. This can be done in several ways (see Simoncini [2003] for a detailed discussion). One possible way is to solve the following modified problem instead of (8),

$$(R + SCHC^* S^*)\dot{\Psi} - k_0^2 S\dot{\Psi} = 0, \tag{9}$$

where the columns of C form a basis of $\ker(R)$ and H is a positive definite matrix so that zero eigenvalues are shifted and do not disturb computations.

This modified eigenproblem can be easily implemented in SLEPc with the aid of shell matrices, in a similar way as the one illustrated in the previous case study.

5.3 Information Retrieval

In the context of information retrieval, many techniques are based on vector space models, thus involving algebraic computations. A matrix is built containing the frequencies (or weights) at which terms appear in the documents. A preprocessing step reduces the size of this terms-by-documents matrix by running a stemming algorithm and also eliminating stop words, that is, terms with low semantic relevance such as articles, conjunctions, etc. Then, a query can be evaluated by computing the cosine of its angle with respect to each document [Berry and Browne 1999].

In techniques such as Latent Semantic Indexing (LSI), the large terms-by-documents matrix is approximated by a low-range matrix so that query evaluation is faster while trying to maintain a similar precision-recall ratio. This approximation can be carried out with a truncated singular value decomposition (SVD). Given a real m by n matrix A, there exist two orthogonal matrices $U \in \mathcal{R}^{m\times m}$ and $V \in \mathcal{R}^{n\times n}$ such that $U^TAV = \text{diag}(\sigma_1, \ldots, \sigma_p)$, with $p = \min\{m, n\}$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$. The values σ_i are called *singular values* while the columns of U and V are the left and right singular vectors, respectively. The truncated SVD is the rank-k partial SVD, where $k < p$, that is, only the first k singular triplets (σ_i, u_i, v_i) are used. The matrix $\tilde{A} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ can be a good approximation of A if the discarded singular values are small compared to the accepted ones.

A direct relation exists between the SVD and the eigendecomposition of matrices A^TA and AA^T. Typically, the smallest one is chosen for computation. An alternative is to compute the eigenpairs of

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \sigma_i \begin{bmatrix} u_i \\ v_i \end{bmatrix}, \tag{10}$$

which is generally best suited for computing the smallest singular values. Again, the three alternatives can be implemented by making use of shell matrices. For instance, in the case of A^TA, the matrix-vector multiply function would simply need to carry out the following operations:

```
MatMult( A, x, w );
MatMultTranspose( A, w, y );
```

This is included as one of the example codes in the SLEPc distribution file. Multiplication by the transpose matrix can cause bad parallel performance, so if memory is not an important limitation a better strategy would be to store A^T explicitly.

5.4 Performance Tests

In order to complement the description of the case studies, this section gives an indication of the computing time in typical uses, as well as some results

Table II. Execution Time (in Seconds), Speedup, and Efficiency (in %)

	Lambda Modes			Cavity Resonators			Info. Retrieval		
p	T_p	S_p	E_p	T_p	S_p	E_p	T_p	S_p	E_p
1	169.38	1.00	100	1044.81	1.00	100	—	—	—
2	100.73	1.68	84	572.81	1.82	91	1490.25	1.00	100
4	60.54	2.80	70	304.85	3.43	86	898.60	1.66	83
8	34.17	4.96	62	161.69	6.46	81	579.18	2.57	64
16	19.14	8.85	55	91.06	11.47	71	453.02	3.29	41

regarding parallel performance in the three cases. All the tests have been run on a cluster of 20 biprocessor nodes with Pentium Xeon processors at 2 GHz with 2 Gbytes of memory and connected by an SCI network with 4×5 torus topology. The tests used up to 16 processors, all in different nodes.

Table II shows the measured wall time (T_p) corresponding to the solution of the eigenvalue problem with p processors, together with the speedup ($S_p = T_1/T_p$) and efficiency ($E_p = S_p/p$). The results of the lambda modes problem correspond to the computation of the dominant mode in a real reactor with associated matrices of order 125320. In the electromagnetism case study, the three largest modes of an empty rectangular box cavity were computed with a finite element mesh of 151890 edges. In both cases, the systems of linear equations were solved by the conjugate gradient method with diagonal preconditioning. Regarding the information retrieval test, the times corresponded to the computation of 300 singular values and vectors of a terms-by-documents matrix of order 221835×226087. The memory requirements were too high for running with one processor, so speedup and efficiency are given relative to two processors.

6. CONCLUSIONS

This article describes SLEPc, a parallel library for the solution of large sparse eigenvalue problems. It is based on PETSc and aims at being able to cope with problems arising in real applications by using techniques such as shift-and-invert transformations and state-of-the-art solvers. It offers a growing number of solution methods as well as interfaces to external eigenvalue packages.

For users that have already worked with PETSc, learning SLEPc is straightforward. With little programming effort it is possible to easily test different solution strategies for a given eigenproblem. Once the problem has been specified, it is extremely easy to experiment with different solution methods and to carry out studies by varying parameters such as the value of the shift.

SLEPc inherits all the good properties of PETSc, including portability, scalability, efficiency, and flexibility. Due to the seamless integration with PETSc, the user has at his or her disposal a wide range of linear equation solvers. In particular, the user holds control over the linear solvers associated with eigenproblems as if they were stand-alone objects.

The use of shell matrices allows the easy formulation of complicated block-structured eigenproblems, as illustrated in the three case studies presented in Section 5. The parallel efficiency achieved with these application codes are representative of the benefits that can be obtained without requiring experience in parallel programming.

The SLEPc distribution file is available for download at the following Web address: `http://www.grycap.upv.es/slepc`.

REFERENCES

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*. SIAM Press, Philadelphia, PA. Available online at `http://www.netlib.org/etemplates`.

BALAY, S., BUSCHELMAN, K., GROPP, W., KAUSHIK, D., KNEPLEY, M., MCINNES, L. C., SMITH, B., AND ZHANG, H. 2004. PETSc users manual. Tech. rep. ANL-95/11, Rev. 2.2.1. Argonne National Laboratory, Argonne, IL.

BERRY, M. W. AND BROWNE, M. 1999. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM Press, Philadelphia, PA.

HERNANDEZ, V., ROMAN, J. E., AND VIDAL, V. 2004. SLEPc users manual. Tech. rep. DSIC-II/24/02—Rev. 2.2.1. D. Sist. Inform. y Comp., Universidad Politécnica de Valencia, Valencia, Spain.

HERNANDEZ, V., ROMAN, J. E., VIDAL, V., VERDU, G., AND GINESTAR, D. 2003. Resolution of the neutron diffusion equation with SLEPc, the Scalable Library for Eigenvalue Problem Computations. In *Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew*. American Nuclear Society, Gatlinburg, TN.

LEHOUCQ, R. B., SORENSEN, D. C., AND YANG, C. 1998. *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. SIAM Press, Philadelphia, PA.

MARQUES, O. A. 1995. BLZPACK: Description and user's guide. Tech. rep. TR/PA/95/30. CERFACS, Toulouse, France.

SIMONCINI, V. 2003. Algebraic formulations for the solution of the nullspace-free eigenvalue problem using the inexact Shift-and-Invert Lanczos method. *Numer. Lin. Algebra Appl. 10*, 357–375.

WU, K. AND SIMON, H. 1997. A parallel Lanczos method for symmetric generalized eigenvalue problems. Tech. rep. LBNL-41284. Lawrence Berkeley National Laboratory, Berkeley, CA.

WU, K. AND SIMON, H. 2001. Thick-restart Lanczos method for large symmetric eigenvalue problems. In *SIAM J. Matrix Anal. Appl. 22*, 2, 602–616.

Received September 2003; revised June 2004; accepted November 2004