Implementation of a variable block Davidson method with deflation for solving large sparse eigenproblems

Article *in* Numerical Algorithms · August 1999

DOI: 10.1023/A:1019199700323 · Source: OAI

CITATIONS
19

READS
313

2 authors, including:

Roger B. Sidje
University of Alabama
76 PUBLICATIONS **1,973** CITATIONS

SEE PROFILE

Numerical Algorithms 20 (1999) 217–240

Implementation of a variable block Davidson method with deflation for solving large sparse eigenproblems

Miloud Sadkane [a] and Roger B. Sidje [b]

[a] *Département de Mathématiques, Université de Bretagne Occidentale, 6, Avenue Le Gorgeu, B.P. 809, F-29285 Brest Cedex, France*
E-mail: sadkane@univ-brest.fr

[b] *Department of Mathematics, University of Queensland, Brisbane, QLD 4072, Australia*
E-mail: rbs@maths.uq.edu.au

The Davidson method is a preconditioned eigenvalue technique aimed at computing a few of the extreme (i.e., leftmost or rightmost) eigenpairs of large sparse symmetric matrices. This paper describes a software package which implements a deflated and variable-block version of the Davidson method. Information on how to use the software is provided. Guidelines for its upgrading or for its incorporation into existing packages are also included. Various experiments are performed on an SGI Power Challenge and comparisons with ARPACK are reported.

Keywords: eigenvalue, Davidson method, Arpack, sparse matrix, preconditioner

AMS subject classification: 65F15, 65F50

1. Introduction

The Davidson method [5] is a preconditioned eigenvalue technique aimed at computing a few of the outer (i.e., leftmost or rightmost) eigenpairs of large sparse symmetric matrices. This method has gained quite an interest in quantum chemistry where it emanated. However, for the classical Davidson method to be suitable, the matrix dealt with must be strongly diagonally dominant (in the sense that its eigenvectors are close to the canonical vectors). The algorithm then uses the diagonal as preconditioner.

Further recent theoretical investigations carried out in Morgan and Scott [17], Crouzeix et al. [3], Sadkane [20], and Sleijpen and Van der Vorst [22] have shown that the method can be used with more general preconditioners and therefore can be applied successfully to matrices arising in other fields. The crux now resides in choosing an appropriate preconditioner for the problem at hand.

This paper describes a software package, called hereafter DAVIDSON, which implements the generalized extension of the Davidson method. The software package is developed in Fortran 77 and its implementation is derived from the considerations studied in [3]. For instance, it is a *block version*. The block implementation increases memory performance through data locality and cache reuse. More specifically, this

implementation is a variable block implementation in the sense that the block-size is adjusted (downward or upward) on the fly. Also, *several eigenpairs* are computed at once and the eigenpairs that converge are *deflated*. The correction stage (or preconditioning – step 7 of the algorithm given in section 2) can be implemented with a *general preconditioner* instead of using only the diagonal as it was proposed originally. A variety of preconditioners are included in the package.

Additionally, the package contains interface routines for input/output management and this allows an end-user to exploit the software directly in a stand-alone fashion without extra coding. Besides, the modular organization of the package enables interested users to either extract the core computational routines in view of utilizing them elsewhere, or add in the present package their own routines for handling matrix-dependent operations. These include: matrix loading, matrix multiplication, preconditioning. A section dealing with these concerns is provided which outlines the necessary steps to follow. In its current distribution, the package is ready-to-use for matrices stored under the Harwell–Boeing storage format (HBO) [6], Compressed Column Storage format (CCS) and the Coordinates storage format (COO) [18]. The implementation has been successfully tested on SUN and SGI platforms.

Other single and block versions of the Davidson method have already been investigated, with numerical results and comparisons of different variants. For instance, Liu [15] and Kosugi [11] have suggested block versions of the Davidson method and a resulting implementation was proposed in Stathopoulos and Fisher [24]. A number of other implementations of the classical Davidson method have been published, see, for example, Weber et al. [25] and Cisneros and Bunge [2]. However, these cited works, unlike the study done here, are confined to a diagonal-type preconditioning. They do not consider general user-supplied preconditioners, nor do they incorporate a variable block and deflation.

The organization of this paper is as follows. In section 2, key issues affecting the implementation are discussed in detail. In section 3, the installation and the utilization of the software are discussed. In section 4, the overall structure of the software is examined. Finally, in section 6 various experiments are performed on an SGI Power Challenge in serial mode and comparisons with the implicitly restarted Arnoldi algorithm ARPACK [12,14,23] are presented.

2. The generalized Davidson algorithm

2.1. Pseudo-code

The following algorithm computes the ℓ leftmost (or rightmost) eigenpairs of a real symmetric matrix A. C_k^i, $i = 1, \ldots,$ stands for a set of preconditioning matrices, *tol* is a user-supplied tolerance, m is a given integer which limits the dimension of the search subspace and MGS stands for the Modified Gram–Schmidt procedure [7].

Algorithm 1. Variable-block Davidson with deflation
Choose an initial block-size n_{b_0} and set $\nu_0 := \max\{n_{b_0}, \ell\}$, $n_b := \nu_0$, *nevf* $:= 0$;
Choose an initial orthonormal matrix $V_1 := [v_1, \ldots, v_{n_b}] \in \mathbb{R}^{n \times n_b}$;
for $k = 1, \ldots$ **do**

1. Compute the matrix $W_k := AV_k$;
2. Compute the Rayleigh matrix $H_k := V_k^{\mathrm{T}} W_k$;
   Set $\nu := \min\{\dim(H_k), \nu_0 - \textit{nevf}\}$;
3. Compute the ν leftmost (or rightmost) eigenpairs $(\lambda_k^i, y_k^i)_{1 \leqslant i \leqslant \nu}$ of H_k;
4. Compute the Ritz vectors $X_k := [x_k^1, \ldots, x_k^\nu]$, where $x_k^i := V_k y_k^i$, $1 \leqslant i \leqslant \nu$;
5. Compute the residuals $R_k := [r_k^1, \ldots, r_k^\nu]$, where $r_k^i := \lambda_k^i x_k^i - W_k y_k^i$, $1 \leqslant i \leqslant \nu$;
   Set n_c to the number of satisfactory residuals;
6. Deflate the n_c eigenpairs that have converged;
   nevf $:=$ *nevf* $+ n_c$; **if** *nevf* $\geqslant \ell$ **then** exit;
7. Correct the unsatisfactory residuals $t_k^i := C_k^i r_k^i$, $n_c + 1 \leqslant i \leqslant \nu$;
8. Select the meaningful corrections $\hat{t}_k^1, \ldots, \hat{t}_k^{n_{\hat{b}}}$, $n_{\hat{b}} \leqslant n_{b_0}$, with algorithm 2;
9. **if** $\dim(V_k) > m - n_{\hat{b}}$ **or** $n_c \neq 0$ **or** $n_{\hat{b}} = 0$ **then** $\{\ldots\ldots\ldots\ldots$ *restart*$\}$
   $V_{k+1} := \mathrm{MGS}(x_k^{n_c+1}, \ldots, x_k^\nu, \hat{t}_k^1, \ldots, \hat{t}_k^{n_{\hat{b}}})$;
   $n_b := \nu - n_c + n_{\hat{b}}$;
   else $\{\ldots\ldots\ldots\ldots\ldots\ldots\ldots$ *expansion of the search subspace*$\}$
   $V_{k+1} := \mathrm{MGS}(V_k, \hat{t}_k^1, \ldots, \hat{t}_k^{n_{\hat{b}}})$;
   $n_b := n_{\hat{b}}$;
   endif

endfor.

2.2. Implementation

Algorithm 1 is implemented in the routine *davson* located in the file *davpack.f*. This section highlights some key details useful for a better understanding of the implementation.

The algorithm caters for block-size greater or smaller than the effective number of desired eigenpairs. Either way may improve performance, especially when the problem is influenced by clustered eigenvalues and/or eigenvalues with high multiplicity. Accommodating any block-size affects the readability and while tracing the algorithm, readers may set n_{b_0} equal to ℓ for convenience. Convergence means that the number of desired eigenpairs is attained – with possible repetition of eigenvalues.

The code makes use of updatings. In steps 1 and 2 we only need to compute the last columns of W_{k+1} and H_{k+1}. Indeed, if at step 8 we let $\widehat{T}_k = [\hat{t}_k^1, \ldots, \hat{t}_k^{n_{\hat{b}}}]$, then to compute $V_{k+1} = [V_k, U_k] = \mathrm{MGS}(V_k, \widehat{T}_k)$ it suffices to compute U_k since V_k is already orthonormal. It then follows that

$$W_{k+1} = AV_{k+1} = [W_k, AU_k];$$

$$H_{k+1} = V_{k+1}^{\mathrm{T}} W_{k+1} = \begin{bmatrix} \boxed{H_k} & V_k^{\mathrm{T}} AU_k \\ (V_k^{\mathrm{T}} AU_k)^{\mathrm{T}} & U_k^{\mathrm{T}} AU_k \end{bmatrix}.$$

In the code, a user-supplied block matrix-multiplication routine is called to compute AU_k after which H_{k+1} is updated by simply computing $V_{k+1}^T AU_k$. Next the computation of the desired eigenpairs of H_{k+1} is carried out (in step 3) using the LAPACK routine *DSYEVX*. Since this routine destroys the lower-portion (including the diagonal) of the matrix H_{k+1}, the code later replicates the strict upper-part of H_{k+1} into its strict lower-part. The diagonal is restored from a separate array in which it is duplicated.

Given that not all eigenpairs converge at the same time, we actually examine each eigenpair individually. The code provides a choice of convergence criteria. The user can set a parameter *anorm* so that if *anorm* > 0, an eigenpair (x, λ) with $\|x\| = 1$, is accepted if the relative residual $\|Ax - \lambda x\|/$*anorm* $\leqslant$ *tol*, where $\|\cdot\|$ denotes the Euclidean norm and *tol* is a user-required accuracy tolerance. If *anorm* $= 1$, the test is therefore based on the absolute residual $\|Ax - \lambda x\| \leqslant$ *tol*. If *anorm* $= 0$, the code uses the relative residual $\|Ax - \lambda x\|/\max(\varepsilon^{2/3}, |\lambda|) \leqslant$ *tol*, where ε is the machine unit roundoff computed in the software (see the subroutine *davson*). If an eigenpair is accepted, it is deflated (step 6). The deflation involves shifting the satisfactory eigenvectors to the beginning of X (the matrix of current approximate eigenvectors). This control is also known as "locking" [8]. For instance, if x_p and x_q, say, are the first two eigenvectors to converge, then X is reshuffled into $X = [x_p, x_q, \ldots]$ and the code later operates with $X(:, 3 : \nu)$ in a convenient manner. The deflation also involves orthogonalizing forthcoming basis vectors against the satisfactory eigenvectors. Thus if *nevf* is the number of eigenvectors found so far, then any basis vector v_j is orthogonalized against $X(:, 1 : \textit{nevf})$ and against $V(:, 1 : j-1)$. We draw the attention of readers to the fact that all vectors which are orthogonalized against the current basis are *always* orthogonalized against the current satisfactory eigenvectors too, although this is not mentioned later in our exposition. Whenever the deflation arises, the code restarts (step 9) using a basis consisting of unsatisfactory eigenvectors $X(:, \textit{nevf} + 1 : \nu)$ and a selection of corrected residuals associated to them (the selection process is described below). At most n_{b_0} candidates (or fewer if there is not enough remaining room in the basis) are retained.

The matrices C_k^i (step 7) are often of the form $C_k^i = (M - \lambda_k^i I)^{-1},\ i = 1, \ldots, \nu$, where M is an approximation of A (approximation in the sense given in [3,20]). In the classical Davidson method, M was taken as the diagonal of A. This package includes a variety of preconditioners and users can also add their own tailor-made preconditioners. Built-in preconditioners are outlined in table 1 where the numeric identifier ID denotes the different preconditioners available in the package and where $x,\ \lambda,\ r,\ t$ denote, respectively, the current Ritz vector, Ritz value, residual vector, and the corrected vector that is added to the basis. These preconditioners are "resilient".

Table 1
Built-in preconditioners. The selection of a preconditioner is done through a numeric identifier (ID).

ID	Meaning Davidson with
0	NC – No Corrector: $t = r$.
1	DC – Diagonal Corrector: $t = (\text{diag}(A) - \lambda I)^{-1} r$.
2	TC – Tridiagonal Corrector: $t = (\text{tridiag}(A) - \lambda I)^{-1} r$.
3	PC – Pentadiagonal Corrector: $t = (\text{pentadiag}(A) - \lambda I)^{-1} r$.
−1	GS – Gauss–Seidel: $t = (A - \lambda I)^{-1} r$ with *one* iteration of GS and a zero starting guess. The implementation was finely-tuned in this respect.
−2	IC – Incomplete Choleski: $t = (A - \lambda I)^{-1} r$ with the square-root free LDL^{T} modification of the code in [9,10]. When several eigenpairs are being sought after (i.e., block mode), only *one* decomposition is performed with λ taken as the mean-value of the current Ritz values. Afterwards, backsolves are applied on each of the residual vectors.
−3	EC – Exponential Corrector: $t = \exp(\tau A)x$ with EXPOKIT [21] using a basis-size of 4 and a tolerance of 10^{-2}. The user can modify these parameters. The scaling-factor τ is selected automatically to reduce the cost of the operation (i.e., only *one* integration step was allowed). Note that $\exp(\tau A)x$ and $\exp(\tau(A - \lambda I))x$ are equivalent correctors.

This means that even when they break down, we do not interrupt the entire Davidson algorithm. For instance consider the diagonal corrector $t = (\text{diag}(A) - \lambda I)^{-1} r$, i.e., $t_i = r_i/(a_{ii} - \lambda)$, $i = 1, \ldots, n$. If an entry $a_{pp} - \lambda$ is nearly zero, we simply skip this entry and set $t_p = r_p$. Likewise, if the current IC is not invertible, we simply set $t = r$.

Algorithm 2: Selection of corrections (ORTHO)
Choose $0 \ll \eta < 1$;
Set $n_{\hat{b}} := 0$; *maxorth* := 2;
for $i := 1 : \nu$
  $k := 0$
  repeat
    old := $\|t_i\|$; $k := k + 1$
    Orthogonalization: $t_i := (I - v_j v_j^{\text{T}}) t_i$ **for** $j := 1, \ldots, m + n_{\hat{b}}$;
  until $k =$ *maxorth* **or** $\|t_i\| > \eta \cdot$ *old*;
  if $\|t_i\| >$ *droptol* **then** {t_i is accepted}
    $n_{\hat{b}} := n_{\hat{b}} + 1$; $\hat{t}_{n_{\hat{b}}} := t_i/\|t_i\|$; $v_{m+n_{\hat{b}}} := \hat{t}_{n_{\hat{b}}}$
  endif
endfor

After correcting the residuals (step 7), a filtering is applied to select at most n_{b_0} meaningful ones (step 8). This step plays a key role in the variation of the block-size. Indeed if $n_{\hat{b}}$ vectors are selected, the current block-size becomes $n_{\hat{b}}$. The selection is carried out by the routine named ORTHO which performs an orthogonalization and a selective reorthogonalization of all the candidate vectors against the current basis vectors and retains those which have a significant component in the complement of the ongoing search subspace. Indeed ORTHO first orthogonalizes each candidate vector

against the basis vectors. Then it possibly applies another MGS sweep (i.e., there is a selective reorthogonalization using the criteria in [4]) and the candidate is retained if its norm is greater than a drop-tolerance *droptol*. When a vector is retained, it is normalized and included in the basis and it subsequently participates in the MGS sweeps against the other candidates. If we let $V = [v_1, \ldots, v_m]$ and $T = [t_1, \ldots, t_\nu]$ the pseudo-code of ORTHO (algorithm 2.2) shows how V is incremented with a selected number of vectors from T. We have set $\eta = 0.1$ and *droptol* $= \varepsilon$ in the actual code and in practice, a further precaution is taken. When we are interested in the rightmost eigenpairs, T is scanned from the last column to the first, and when we are interested in the leftmost eigenpairs, T is scanned from the first column to the last (as in the pseudo-code). This method contributes in ensuring that the most meaningful candidates are retained first. The routines ORTHO and MGS are located in the file *davpack.f*.

The Davidson algorithm restarts (step 9) when either of the following situations occur:

(i) the maximum allowable size of the basis is reached,

(ii) no corrected residual is retained by ORTHO, and also

(iii) whenever an eigenpair converges.

Recall that this latter case induces a deflation. Upon restart, the block-size becomes the effective number of vectors in the basis. Hence, if the algorithm started initially with n_{b_0} vectors and all the corrected residuals are retained, the block-size will become $\max\{n_{b_0}, \ell\} + n_{b_0}$. This extreme situation reflects the maximum value that the block-size can attain. However, it is a short-lasting circumstance which ceases as soon as the basis is incremented once more. This is well illustrated by numerical experiments (figures 2, 3 in section 6). The trace of the block-size shows regular impulses at restart positions. The user can print a detailed history of the block-size by setting the *infolevel* parameter to 2. More details about this parameter are given in the file *init.data.example*.

3. Installation and utilization

3.1. Installation

The package is written in Fortran 77 and uses external routines from the well-known dense linear algebra libraries BLAS and LAPACK [1]. Once the compressed package containing the software has been retrieved and unpacked, typing *make alone* builds a simple self-contained stand-alone driver, and typing *make runme* builds a more elaborate driver that uses external data. Once the executables have been successfully built, they can be run directly. They will use default settings. Section 3.2 details input/output data. If the compilation was unsuccessful, one may need to edit the Make-file in order to change either the compiler name or the libraries with respect to the local installations. The driver *alone* is aimed at illustrating how the core computational

routines can be used in isolation. The advanced driver *runme* uses a configurable environment that enables the user to repeat and study the computations under different settings (e.g., different preconditioners) and explore several eigenproblems. From now on, the description applies to the driver *runme*.

3.2. Managing input/output

The driver *runme* uses a number of parameters that can be tuned by the user. All these parameters and their default values are summarized in table 2.

To override a default value one passes through a file referred to as *init.data* which must be created by the user (see table 3). All customizations should be grouped in this file according to a simple rule which we now describe. To modify the value of the parameter *name* as given in table 2, it suffices to include in the file *init.data* two lines as follows:

```
name:
value
```

For example, to use a basis of size 30 instead of the default size of 40, it suffices to include the two lines:

```
basis:
30
```

Table 2
Input parameters. An expanded version of this table is given in the README file and further explanations are given in the file *init.data.example* as well.

Name	Description	Default	Example
matfile	matrix file name	screen	*gr3030*
mattype	matrix storage type	*hbo*	*coo*
guessfile	initial guess file (for starting eigenvectors) if unspecified, guesses are generated internally if specified, guesses are read from the file	unspecified	*guess_file*
errfile	file for error messages	screen	*error_file*
outfile	file for output results	screen	*output_file*
outfmt	format for output results	1P, E22.15	1P, E11.3
infolevel	information level gauge	1	3
basis	basis-size	40	30
block	block-size	1	4
eigenpair	number of desired leftmost/rightmost eigenpairs – ℓ searches for the ℓ leftmost eigenpairs	1	-3
iteration	maximum allowable number of iterations	100	50
corrector	type of corrector (preconditioner)	1	3
outmax	number of rows written on output – k causes the last k rows to be written	5	-7
tolerance	required tolerance on output results	1.0E$-$7	1.0E$-$5

Table 3
An example of *init.data*.

```
# Silent running!
infolevel:
0

# Set the matrix filename and type
matfile:
gr3030
mattype:
hbo

# Look for the 3 smallest eigenpairs using a block-size of 3.
block:
3
eigenpair:
-3

# Pentadiagonal correction
corrector:
3
```

Table 4
Output obtained with the above *init.data*. This problem is merely illustrative and has no particular/intrinsic implication. The spectrum as computed by LAPACK itself (when the matrix is made dense for the purpose of the experiments) is such that $\lambda_1 = 0.0614628239 \leqslant \lambda_2 = \lambda_3 = 0.1531843111 \leqslant \lambda_4 = 0.2439646117 \leqslant \lambda_5 = \lambda_6 = 0.3050073347 \leqslant \cdots \leqslant \lambda_{897} = \lambda_{898} = 11.9286959239 \leqslant \lambda_{899} = \lambda_{900} = 11.9590598825$.

```
Runtime (seconds): 3.010E+00
Total number of iterations used: 9
Total number of matrix * vector: 315

RESIDUALS =
  9.956196589564992E-08 7.243564772965687E-08 9.527277513806256E-08

EIGENVALUES =
  6.146282392742993E-02 1.531843111273325E-01 1.531843111273338E-01

EIGENVECTORS =
  6.603244615201538E-04 -5.602146448759126E-04 1.771633259792640E-03
  1.313873146434165E-03 -1.078911591320393E-03 3.506504760921992E-03
  1.953939709654315E-03 -1.516463696194206E-03 5.168990767388935E-03
  2.573956184026209E-03 -1.836945055162193E-03 6.725700605953140E-03
  3.167560361385945E-03 -2.009787164787331E-03 8.146488023314377E-03
```

These lines must start at the first column. The colon (`:`) is required, for it delimits the headword which specifies the name of the parameter. A line whose headword does not match any of the expected names is considered as a comment line. Table 3 shows a possible content of the file *init.data*. Users can request a different initialization file by typing *runme initfile*, where *initfile* denotes the name of the initialization file to be used. If the file *init.data* does not exist and there is no initialization file specified on the command line, the code will use default values (table 2) and will prompt the user for the name of a file containing a Harwell–Boeing matrix.

The file *gr3030* used in the example in table 3 comes from the Harwell–Boeing collection. The value of *corrector* specifies which type of preconditioner is to be used. Users can select other built-in preconditioners by using IDs given in table 1 or they can incorporate their own preconditioners in *correc.f* by using extra IDs (see section 5.2 and the README file).

4. Overview of the package

As portrayed in figure 1, the organization of the package was made modular. The figure indicates that *matvec* and *correc* are external subroutines needed by *davpack*. There are three phases, each involving a certain amount of routines:

INPUT PHASE. The modules involved in this phase are located in *io.f*, *randm.f*, and *getmat.f*. The top-level routine *input* in *io.f* calls auxiliary routines that initialize the parameters using either default settings or assignments in *init.data* (or another

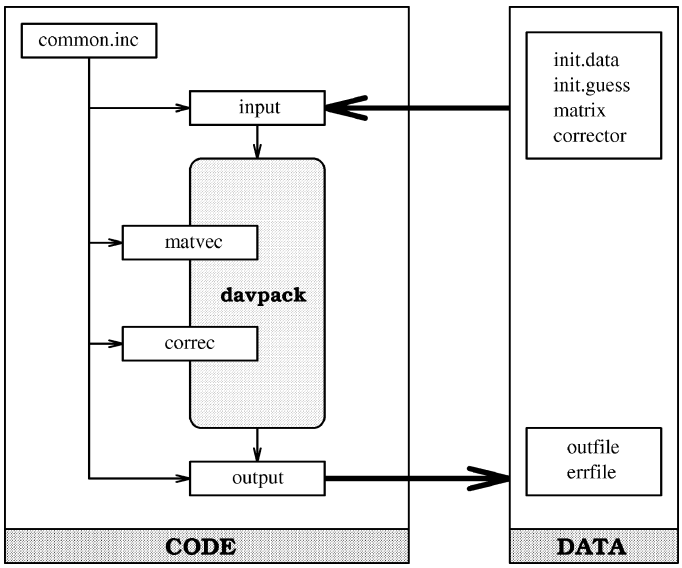

Figure 1. Modular organization.

Table 5
Main files.

Driver	
runme.f	driver using external data
Input/output phases	
common.inc	common/global variables
io.f	input/output management
randm.f	generates a matrix of random numbers with DLARN (LAPACK)
getmat.f	gets the matrix, makes up the corrector, gets initial guesses
eigsrt.f	sorts the eigenpairs with respect to eigenvalues or residuals
Computational phase	
davpack.f	Davidson's algorithm
matvec.f	matrix–vector multiplication
correc.f	routines to perform the correction/preconditioning step
ichol.f	LDL^T modification of the Incomplete Choleski code in [9,10]
expokit.f	relevant subset extracted from the matrix exponential package [21]

initialization file specified by the user). The matrix is loaded and the corrector is made ready. If a *guessfile* has been specified, ℓ initial approximate eigenvectors are read therein (assuming the standard matrix-wise tabulation) – *randm.f* contains a routine that can be used to generate or augment a set of initial guesses.

COMPUTATIONAL PHASE. The modules involved in this phase are located in *davpack.f*, *matvec.f*, *correc.f*, *ichol.f*, *expokit.f*. Using data provided at the input phase, the top-level routine *davson* in *davpack.f* works in conjunction with the matrix–vector and corrector routines to search the desired eigenpairs.

OUTPUT PHASE. The modules involved in this phase are located in *eigsrt.f* and *io.f*. The results returned by the computations are in no particular order. Thus at this phase, the results are sorted (with a call to the routine *eigsrt*) and reported (with a call to the routine *output* in *io.f*). The recipient of the results can be a file whose name was specified under the parameter *outfile*. Upon exit, the file defined by the parameter *errfile* may contain informative or error messages.

The routines are assembled into files as described in table 5. The file *common.inc* is of a particular importance. It contains variables which are *"visible" in all routines apart from that in davpack.f*. Indeed the variables there are carried along using the Fortran "common statement". It follows that these variables can be viewed as *global variables*. To sustain awareness and avoid misuse, we have prefixed these variables with "c_". For instance, c_n is used internally as the order of the matrix, c_nz is the number of nonzero elements, and so forth.

5. Information for the application developer

Lehoucq and Scott [13] have evaluated several general purpose unsymmetric eigensolvers in terms of such criteria as user interface, storage, performance, stability and reliability. As we see, the user interface is a chief issue when designing an

eigensolver, and sometimes it even determines whether the solver will be used at all. Efforts have been devoted on the user interface of the DAVIDSON package. However, although an end-user could typically run the software as it is, an application developer will probably need to implement additional functionality. Possible options are:

1. To extract the core computational routines of this package in view of their utilization from within another existing package.
2. To incorporate into this package additional routines for loading a matrix stored under a particular storage scheme, for efficient matrix–vector multiplication, and for the correction stage.

We shall address both of these options in turn.

5.1. Extracting the core routines

The core computational routines are the internal routines given in table 6. They are grouped altogether in the file *davpack.f*. This file is computationally self-contained and does not use common variables. The routines there are matrix-free, i.e., they are independent of the matrix data storage format. Utilization of these routines is possible outside of the actual package provided the user:

- supplies a matrix–vector multiplication routine (MATVEC),
- supplies a correction routine (CORREC),
- links with BLAS and LAPACK.

The relevant subsets of BLAS and LAPACK needed by the package are included in the files *blas.f* and *lapack.f* for users who do not have the local installations of these libraries. Guidelines for using or not using them are provided in the *Makefile*. The package includes a minimal stand-alone program referred to as *alone.f* which illustrates the use of the core computational routines in isolation.

Table 6
Full list of subroutines directly involved in *davpack.f*. Other auxiliary routines of BLAS and LAPACK are indirectly called from these.

Supplied	Internal	External	
by user		BLAS	LAPACK
MATVEC	DAVSON	DGEMM	DSYEVX
CORREC	MGS	DAXPY	
	ORTHO	DSCAL	
		DDOT	
		DNRM2	
		DCOPY	
		DSWAP	
		IDAMAX	

5.2. Adding user-supplied matrix-dependent routines

One difficulty in designing general purpose sparse matrix packages resides in the existence of several sparse data storage formats. However, this variety is understandable because formats arise naturally when modeling certain problems. This package currently deals only with the Harwell–Boeing, Compressed Column Storage, and Coordinates formats. Users may wish to handle other formats.

There are three possible ways to proceed:

1. Convert the matrix into a format which is used in the package.
2. Extract the matrix-free computational routines and proceed as shown in section 5.1.
3. Adapt the package so that it takes into account other storage schemes. In this case, the files *getmat.f*, *matvec.f*, *correc.f* should be modified. To this end, instructions are given in the headers of these files and the README file contains a detailed guide which, amongst other things, highlights the variables in the file *common.inc* that may be used.

6. Numerical experiments

Extensive experiments were conducted on a SGI Power Challenge in serial mode using 24 matrices from the Harwell–Boeing collection (table 7). The DAVIDSON code as well as the installation of ARPACK were made with the highest level of optimization, i.e., -O3-OPT:IEEE_arithmetic $= 1$. The flag -OPT:IEEE_arithmetic $= 1$ is added to instruct the compiler not to do optimizations which produce less accurate results than required by the IEEE-754 standard. We search for the 5 rightmost/leftmost eigenpairs, with tolerance *tol* $= 10^{-10}$, basis-size $m = 25$, and *itmax* $= 200$ restarts. A method is said to succeed if *all* of the desired eigenpairs are found. It is said to fail otherwise. In case of partial failure, we show *nconv*, the number of convergent eigenpairs attained. In case of total failure, *nconv* $= 0$, and in brief we show dashes (–).

We have set *anorm* $= 0$ in DAVIDSON and, hence, an eigenpair (x, λ) is accepted if the relative residual $\|Ax - \lambda x\|_2 / \max(\varepsilon^{2/3}, |\lambda|) \leqslant$ *tol*, where ε is the machine unit roundoff. This coincides with the stopping criteria built in ARPACK. However, unlike DAVIDSON, the residuals in ARPACK are estimated and, albeit rarely, they may sometimes be deceptive. Therefore after exiting from ARPACK we have computed the actual relative residuals and these are the values reported in the tables. It is seen in table 11 that bcsstm04 and bcsstm06 have not converged properly.

For a given problem, a particular tuning of block-size and corrector produces the best time (table 9). Experiments were done with a block-size ranging from 1 to 7 and only the best runs have been retained for further analysis. In general, these best runs come from a block-size larger than 1, as also noted in Kosugi [11]. It is also seen in table 10 that the best runs are often achieved with GS (Gauss–Seidel corrector) or IC (Incomplete Choleski corrector), thus suggesting that these correctors are superior to

Table 7
Matrices used in the experiments: n is the order and nz is the number of nonzero elements in the half-part (since we are dealing with symmetric matrices). The spectrum $\{\lambda_1, \ldots, \lambda_n\}$ is assumed ordered algebraically as $\lambda_1 \leqslant \cdots \leqslant \lambda_n$.

	Matrix	n	nz	Density (%)	$\|\cdot\|_\infty$	λ_1	λ_n
1	1138bus	1138	2596	0.20	40000.00	3.52E−03	3.01E+04
2	662bus	662	1568	0.36	4000.00	5.05E−03	4.01E+03
3	bcsstk01	48	224	9.72	3570948074.70	3.42E+03	3.02E+09
4	bcsstk02	66	2211	50.76	22286.72	4.21E+00	1.82E+04
5	bcsstk05	153	1288	5.50	6045977.49	4.34E+02	6.20E+06
6	bcsstk08	1074	7017	0.61	76666719997.01	2.95E+03	7.66E+10
7	bcsstk09	1083	9760	0.83	70340029.18	7.10E+03	6.76E+07
8	bcsstk10	1086	11578	0.98	34363670.27	8.54E+01	4.47E+07
9	bcsstk11	1473	17857	0.82	741314969.35	2.96E+00	6.56E+08
10	bcsstk12	1473	17857	0.82	741314969.35	2.96E+00	6.56E+08
11	bcsstk19	817	3835	0.57	207448373984209.00	1.43E+03	1.92E+14
12	bcsstk27	1224	28675	1.91	2919527.38	1.44E+02	3.47E+06
13	bcsstm04	132	132	0.76	0.17	0.00E+00	1.73E−01
14	bcsstm06	420	420	0.24	7602.41	2.20E−03	7.60E+03
15	bcsstm10	1086	11589	0.98	27496.34	−2.06E+03	2.88E+04
16	bcsstm12	1473	10566	0.49	14.90	2.12E−05	1.34E+01
17	bcsstm27	1224	28675	1.91	3298.97	−5.96E+00	3.93E+03
18	gr3030	900	4322	0.53	12.00	6.15E−02	1.20E+01
19	lunda	147	1298	6.01	218151766.59	8.00E+01	2.24E+08
20	nos2	957	2547	0.28	118579200000.00	3.08E+01	1.57E+11
21	nos3	960	8402	0.91	515.89	1.83E−02	6.90E+02
22	nos4	100	347	3.47	0.74	5.38E−04	8.49E−01
23	nos5	468	2820	1.29	555660.00	5.29E+01	5.82E+05
24	nos7	729	2673	0.50	9000000.00	4.15E−03	9.86E+06

the others which are more specialized. For instance, the exponential corrector is suited for the rightmost eigenvalues in the case where the spectrum lies in the real negative axis (see [16]) whereas a diagonal corrector is more efficient if the matrix is strongly diagonally dominant as we have already mentioned. A comparison of the best runs against ARPACK is made in table 11. The factors involved in these runs are further analyzed in related tables. Other straight comparisons are displayed in tables 14 and 15 in which the size of the basis is 30 and 40, respectively. The indications hereafter pertain to the case when the size of the basis is 25.

To measure the reliability of the algorithms and attest whether eigenvalues were not missed, a check of correctness was also carried out (tables 12, 13). Cases of missing multiplicities are indicated with $\bullet\bullet\ \triangleright$ and the faulty (or "misplaced") eigenvalue is underlined. The eigenvalues used as reference (table 8) were computed with the built-in function *eig* of MATLAB which internally switches to the symmetric QR method in our context. Because of rounding errors and the tolerance required on DAVIDSON and ARPACK in the experiments, we view eigenvalues that match up to 10 significant digits as identical.

Table 8
The five distinct rightmost/leftmost eigenvalues with their multiplicities (in superscript). The numbering matches with matrices in table 7. These eigenvalues were computed by MATLAB's built-in function *eig* which in our context uses the symmetric QR variant.

	Reference for the five distinct rightmost eigenvalues				
1	.2105105115E+05[1]	.2194783633E+05[1]	.3000130387E+05[1]	.3001049004E+05[1]	.3014879442E+05[1]
2	.1422994613E+04[1]	.1424742094E+04[1]	.1617986726E+04[1]	.2220033359E+04[1]	.4008113925E+04[1]
3	.2018372795E+10[1]	.2207957140E+10[1]	.2220593407E+10[1]	.2970424445E+10[1]	.3015179090E+10[1]
4	.1438284448E+05[1]	.1511295789E+05[1]	.1621278900E+05[1]	.1665103995E+05[1]	.1822574862E+05[1]
5	.5255490753E+07[1]	.5463067787E+07[1]	.5808326738E+07[1]	.5808726611E+07[1]	.6197287056E+07[1]
6	.1686207543E+11[1]	.2218776454E+11[1]	.2711507179E+11[1]	.4416405745E+11[1]	.7657033866E+11[1]
7	.6654597896E+08[1]	.6654598161E+08[1]	.6679820601E+08[1]	.6720297873E+08[2]	.6760303645E+08[1]
8	.4346427154E+08[1]	.4393949521E+08[1]	.4431269100E+08[1]	.4458118573E+08[1]	.4474305592E+08[1]
9	.6371100351E+09[2]	.6538718159E+09[2]	.6550590910E+09[2]	.6550590910E+09[2]	.6556063155E+09[2]
10	.6371100351E+09[2]	.6538718159E+09[2]	.6550590910E+09[2]	.6550590910E+09[2]	.6556063155E+09[2]
11	.1447913203E+15[1]	.1560653679E+15[2]	.1911085254E+15[2]	.1916338492E+15[2]	.1922160604E+15[2]
12	.3035552754E+07[1]	.3128789997E+07[1]	.3231908139E+07[1]	.3344127105E+07[1]	.3465059451E+07[1]
13	.9213858051E−01[6]	.1254266385E+00[6]	.1379957380E+00[6]	.1413083415E+00[6]	.1728285735E+00[6]
14	.7004440505E+04[12]	.7028218323E+04[1]	.7028218361E+04[1]	.7602410190E+04[4]	.7602410228E+04[4]
15	.2813595337E+05[1]	.2828847122E+05[1]	.2843798632E+05[1]	.2868310004E+05[1]	.2883090837E+05[1]
16	.1244244719E+02[2]	.1259191040E+02[2]	.1319142097E+02[4]	.1325744318E+02[2]	.1341559058E+02[2]
17	.3460249067E+04[1]	.3558168823E+04[1]	.3669122473E+04[1]	.3792212298E+04[1]	.3927163488E+04[1]
18	.1183791562E+02[2]	.1186733840E+02[2]	.1187843564E+02[2]	.1192869592E+02[2]	.1195905988E+02[2]
19	.2122131218E+09[1]	.2165941433E+09[1]	.2197883625E+09[1]	.2210402147E+09[1]	.2238540644E+09[1]
20	.1571916938E+12[1]	.1572257837E+12[1]	.1572523014E+12[1]	.1572712445E+12[1]	.1572826110E+12[1]
21	.6720400662E+03[1]	.6758195632E+03[1]	.6772072368E+03[1]	.6845858636E+03[1]	.6899039606E+03[1]
22	.7954146747E+00[1]	.7979687852E+00[1]	.8158840147E+00[1]	.8367012217E+00[1]	.8491377838E+00[1]
23	.5090933789E+06[1]	.5279354620E+06[1]	.5364439368E+06[1]	.5612720376E+06[1]	.5820291156E+06[1]
24	.6836226639E+07[2]	.6836227287E+07[1]	.8224301537E+07[1]	.8224302770E+07[2]	.9864030300E+07[1]
	Reference for the five distinct leftmost eigenvalues				
1	.3516860006E−02[1]	.9862234734E−01[1]	.1241279307E+00[1]	.1768149305E+00[1]	.1831768532E+00[1]
2	.5047169019E−02[1]	.1623680950E+00[1]	.2638129026E+00[1]	.3351872373E+00[1]	.3658843786E+00[1]
3	.3417267563E+04[1]	.8970009818E+04[1]	.1083565548E+05[1]	.2232699142E+05[1]	.5163408924E+05[1]
4	.4214073733E+01[1]	.4300382397E+01[1]	.5258221526E+01[1]	.2636205495E+02[1]	.3805932197E+02[1]
5	.4339489605E+03[1]	.4430684545E+03[1]	.1442783766E+04[1]	.3226074854E+04[1]	.4151646003E+04[1]
6	.2946410519E+04[1]	.3494108138E+04[1]	.3539629915E+04[1]	.3643714454E+04[1]	.3805034584E+04[1]
7	.7102229057E+04[1]	.2734155790E+05[2]	.5495682031E+05[1]	.7857362575E+05[1]	.7958081381E+05[1]
8	.8535085715E+02[1]	.8548309432E+02[1]	.9500642237E+02[1]	.1036665102E+03[1]	.1167622802E+03[1]
9	.2964059196E+01[1]	.2965967441E+01[1]	.1076627628E+02[1]	.1098851091E+02[1]	.2039041618E+02[1]
10	.2964059196E+01[1]	.2965967441E+01[1]	.1076627628E+02[1]	.1098851091E+02[1]	.2039041618E+02[1]
11	.1434350946E+04[1]	.1612674950E+04[1]	.6469992415E+04[1]	.8181055931E+04[1]	.1296686855E+05[1]
12	.1435646443E+03[1]	.1456035900E+03[1]	.1474851421E+03[1]	.1491779411E+03[1]	.1506381265E+03[1]
13	.0000000000E+00[66]	.1974693867E−01[3]	.2317061005E−01[3]	.3059318841E−01[6]	.5332089274E−01[6]
14	.2199084150E−02[20]	.1176958507E+01[6]	.1201430126E+01[18]	.1852666239E+01[3]	.1891155582E+01[9]
15	−.2060531292E+04[1]	−.2037786261E+04[1]	−.2000370305E+04[1]	−.1949035405E+04[1]	−.1884824806E+04[1]
16	.2118712112E−04[1]	.2118712115E−04[1]	.2136055145E−04[1]	.2136055147E−04[1]	.2136090589E−04[1]
17	−.5958597977E+01[1]	−.5958597963E+01[1]	−.5198055485E+01[1]	−.5198055064E+01[1]	−.4488746706E+01[1]
18	.6146282393E−01[1]	.1531843111E+00[2]	.2439646117E+00[1]	.3050073347E+00[2]	.3942297256E+00[2]
19	.8003510930E+02[1]	.1976505467E+04[1]	.1996764780E+04[1]	.6354111204E+04[1]	.1283833070E+05[1]
20	.3084226603E+02[1]	.1233660915E+03[1]	.1392471275E+03[1]	.2775625587E+03[1]	.4934168059E+03[1]
21	.1828839439E−01[1]	.2488598861E+00[1]	.2671801822E+00[1]	.1166601955E+01[1]	.1939512306E+01[1]
22	.5379528369E−03[1]	.3860235311E−02[1]	.4546036376E−02[1]	.1012314149E−01[1]	.1417578665E−01[1]
23	.5289948260E+02[1]	.6754297175E+02[1]	.1155911521E+03[1]	.1315636348E+03[1]	.1857169377E+03[1]
24	.4154132324E−02[1]	.1033441946E−01[1]	.1197176140E−01[1]	.1342134128E−01[1]	.1504096094E−01[1]

Table 9
Sample timing results for the five rightmost/leftmost eigenpairs. The arrows point to the best block-size and the corresponding best time (in seconds) is underlined. These results highlight the fact that all experiments were conducted with a block-size ranging from 1 to 7. Out of all the timings, we only retained the best runs for further analysis (see table 10 below). The nmult column under EC shows the number of matrix–vector products used by the DAVIDSON routine only, and between parentheses, the aggregate number of matrix–vector products used by both DAVIDSON and the exponential corrector. A method is said to succeed if all of the desired eigenpairs are found. It is said to fail otherwise. In case of partial failure, we show *nconv*, the number of convergent eigenpairs attained. In case of total failure, $nconv = 0$, and for brevity we show dashes (–).

bcsstk08		NC		DC		GS		IC		EC	
block-size		nmult	time	nmult	time	nmult	time	nmult	time	nmult	time
Rightmost	1	60	0.25	77	0.49	72	0.63	74	6.08	60(745)	1.73
	2	58	0.21	100	0.46	84	0.49	83	3.46	58(453)	1.06
⟶	3	67	**0.20**	103	0.41	98	0.49	84	2.70	67(397)	0.92
	4	77	0.23	127	0.45	108	0.48	96	2.42	77(402)	0.93
	5	83	0.24	136	0.46	133	0.52	109	2.23	83(398)	0.93
	6	93	0.28	157	0.57	159	0.68	124	2.43	93(478)	1.13
	7	115	0.33	188	0.65	174	0.72	153	2.43	115(525)	1.24
Leftmost	1	–	–	2367	16.07	1009	8.88	1159	107.80		
	2	–	–	3197	15.79	1102	6.65	1171	62.15		
⟶	3	–	–	3630	15.83	1151	**5.93**	1458	55.71		
	4	–	–	–	–	1564	7.21	1512	45.05	N/A	
	5	–	–	–	–	1877	7.77	2686	56.83		
	6	–	–	–	–	2146	9.51	2622	59.79		
	7	–	–	–	–	3212	13.89	3719	61.66		

The history of convergence of some representative best runs is also illustrated graphically (figures 2 and 3). The relative residual (curve-like) and the block-size (histogram-like) are plotted against the number of matrix–vector multiplications. The residual (representing the maximum of all relative residuals) is associated with the left y-axis while the block-size is associated with the right y-axis. As hinted in section 2.2, impulses of the block-size occur at restart positions. As eigenpairs are found, the block-size diminishes accordingly.

From the tables, one can notice for these data sets that both DAVIDSON and ARPACK appear more successful in computing the rightmost eigenpairs than the leftmost ones.

Although limited, multiplicity or clustering of eigenvalues may cause a spurious behavior of either algorithm, e.g., bcsstk11, bcsstk12 and bcsstm06 in table 12 for ARPACK; bcsstm04, bcsstm06, and lunda in table 13 for DAVIDSON. The fault can range from a simple miss of multiplicity to a total miss of the spectrum's portion being sought after. When reliability is at stake, it would be preferable to repeat the computations with different settings. The DAVIDSON code, for instance, can be run with different basis-size, block-size, and corrector. It has been our ob-

Table 10
Summary of the parameters of DAVIDSON for the best runs when we search for the five rightmost/leftmost eigenpairs, with tolerance $tol = 10^{-10}$, basis-size $m = 25$.

	Five rightmost eigenpairs				Five leftmost eigenpairs			
	nmult	time	block-size	corrector	nmult	time	block-size	corrector
1138bus	132	0.38	3	NC	$nconv = 3$		1	IC
662bus	97	0.22	3	NC	1474	6.13	3	IC
bcsstk01	99	0.09	5	GS	255	0.22	7	GS
bcsstk02	134	0.14	5	GS	483	0.61	6	GS
bcsstk05	165	0.18	4	NC	765	1.58	2	GS
bcsstk08	67	0.20	3	NC	1151	5.93	3	GS
bcsstk09	789	3.86	6	GS	$nconv = 4$		2	GS
bcsstk10	267	1.24	2	NC	–	–	–	–
bcsstk11	228	2.34	1	DC	–	–	–	–
bcsstk12	369	3.70	1	DC	–	–	–	–
bcsstk19	166	0.74	2	NC	–	–	–	–
bcsstk27	192	1.24	2	NC	4542	70.22	1	GS
bcsstm04	65	0.05	5	IC	39	0.10	1	GS
bcsstm06	129	0.18	5	IC	107	0.17	4	IC
bcsstm10	501	2.37	2	NC	498	2.73	4	GS
bcsstm12	237	2.20	2	GS	–	–	–	–
bcsstm27	191	1.24	2	NC	3186	26.41	6	GS
gr3030	681	3.14	2	GS	297	1.65	5	IC
lunda	222	0.29	4	GS	186	0.22	5	GS
nos2	307	1.19	5	IC	–	–	–	–
nos3	331	1.33	2	NC	396	5.71	4	IC
nos4	227	0.24	4	GS	204	0.21	5	IC
nos5	175	0.40	4	GS	2232	6.25	3	GS
nos7	125	0.31	3	NC	–	–	–	–

servation that this allows detecting discrepancies and acquiring a certain degree of confidence.

We observe that ARPACK is often faster than DAVIDSON when computing the rightmost eigenpairs whereas in the leftmost case the opposite situation happens. The matrices are of very low density (table 7) and as Morgan and Scott [17] have pointed out, the overhead in the Davidson algorithm is not cost effective if the formation of the matrix–vector product is very cheap. This explains why Davidson with NC (No Corrector) behaves so well in some instances – it is that the overhead in the other correctors has not paid off.

7. Conclusion

We have presented a Fortran 77 implementation of a variable block version of the Davidson method for computing the extreme eigenpairs of large symmetric matrices. The resulting package is a self-contained suite built upon the libraries BLAS and LAPACK. The package can be used directly with a number of built-in preconditioners

Table 11
Best CPU time and relative residuals for the five rightmost/leftmost eigenpairs.

Rightmost	DAVIDSON		ARPACK		DAVIDSON relative residuals					ARPACK relative residuals				
	nmult	time	nmult	time										
1138bus	132	0.38	66	0.20	E−11	E−11	E−11	E−11	E−11	E−13	E−15	E−15	E−16	E−15
662bus	97	0.22	48	0.09	E−11	E−10	E−11	E−10	E−11	E−14	E−14	E−15	E−15	E−15
bcsstk01	99	0.09	48	0.03	E−11	E−11	E−12	E−11	E−11	E−16	E−16	E−16	E−15	E−15
bcsstk02	134	0.14	50	0.04	E−11	E−11	E−11	E−11	E−11	E−14	E−15	E−16	E−16	E−15
bcsstk05	165	0.18	67	0.06	E−11	E−11	E−11	E−11	E−11	E−15	E−15	E−15	E−16	E−16
bcsstk08	67	0.20	30	0.09	E−11	E−10	E−11	E−12	E−11	E−14	E−15	E−16	E−15	E−15
bcsstk09	789	3.86	247	1.03	E−11	E−11	E−11	E−11	E−11	E−15	E−15	E−12	E−15	E−15
bcsstk10	267	1.24	108	0.46	E−10	E−11	E−11	E−11	E−11	E−11	E−13	E−12	E−15	E−15
bcsstk11	228	2.34	183	1.08	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−12	E−13	E−13
bcsstk12	369	3.70	183	1.07	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−12	E−13	E−13
bcsstk19	166	0.74	83	0.22	E−11	E−11	E−11	E−11	E−11	E−12	E−15	E−11	E−14	E−12
bcsstk27	192	1.24	85	0.53	E−10	E−11	E−11	E−11	E−11	E−13	E−15	E−16	E−15	E−16
bcsstm04	65	0.05	74	0.05	E−14	E−14	E−15	E−14	E−14	E−15	E−16	E−17	E−16	E−15
bcsstm06	129	0.18	140	0.18	E−12	E−15	E−12	E−15	E−15	E−15	E−15	E−14	E−15	E−14
bcsstm10	501	2.37	178	0.78	E−11	E−11	E−11	E−11	E−11	E−12	E−15	E−15	E−16	E−15
bcsstm12	237	2.20	124	0.63	E−11	E−11	E−11	E−10	E−11	E−12	E−12	E−11	E−13	E−15
bcsstm27	191	1.24	85	0.53	E−11	E−10	E−11	E−11	E−10	E−11	E−15	E−15	E−15	E−15
gr3030	681	3.14	354	1.05	E−11	E−11	E−11	E−11	E−11	E−13	E−11	E−15	E−15	E−15
lunda	222	0.29	121	0.11	E−10	E−11	E−11	E−10	E−11	E−13	E−15	E−15	E−15	E−16
nos2	307	1.19	1021	3.09	E−11	E−11	E−11	E−11	E−11	E−10	E−11	E−14	E−15	E−15
nos3	331	1.33	143	0.53	E−11	E−10	E−11	E−11	E−11	E−13	E−14	E−14	E−15	E−15
nos4	227	0.24	105	0.07	E−10	E−11	E−11	E−11	E−11	E−13	E−13	E−16	E−16	E−16
nos5	175	0.40	102	0.18	E−11	E−11	E−11	E−11	E−11	E−13	E−16	E−15	E−16	E−16
nos7	125	0.31	65	0.17	E−11	E−11	E−10	E−11	E−11	E−16	E−16	E−15	E−16	E−16
Leftmost	DAVIDSON		ARPACK		DAVIDSON relative residuals					ARPACK relative residuals				
	nmult	time	nmult	time										
1138bus	$nconv = 3$		–	–	E−11	E−11	E−11	E−05	E−01	–	–	–	–	–
662bus	1474	6.13	3179	6.84	E−11	E−10	E−10	E−10	E−10	E−10	E−11	E−12	E−12	E−11
bcsstk01	255	0.22	3918	2.29	E−11	E−11	E−11	E−11	E−11	E−10	E−10	E−10	E−11	E−10
bcsstk02	483	0.61	411	0.34	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−12	E−13	E−12
bcsstk05	765	1.58	785	0.76	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−12	E−11	E−11
bcsstk08	1151	5.93	4530	17.72	E−11	E−11	E−11	E−11	E−11	–	–	–	–	–
bcsstk09	$nconv = 4$		975	4.17	E−11	E−11	E−11	E−11	E−06	E−12	E−11	E−12	E−12	E−13
bcsstk10	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bcsstk11	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bcsstk12	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bcsstk19	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bcsstk27	4542	70.22	–	–	E−10	E−10	E−11	E−11	E−11	–	–	–	–	–
bcsstm04	39	0.10	131	0.09	E−11	E−11	E−15	E−15	E−15	E−06	E−06	E−07	E−06	E−09
bcsstm06	107	0.17	1956	2.55	E−14	E−11	E−11	E−11	E−11	E−09	E−10	E−09	E−09	E−09
bcsstm10	498	2.73	298	1.31	E−11	E−11	E−11	E−11	E−11	E−15	E−15	E−15	E−14	E−12
bcsstm12	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bcsstm27	3186	26.41	2619	17.30	E−11	E−10	E−11	E−11	E−11	E−13	E−13	E−13	E−13	E−11
gr3030	297	1.65	210	0.61	E−10	E−11	E−11	E−11	E−11	E−14	E−12	E−11	E−14	E−11
lunda	186	0.22	3819	3.62	E−10	E−11	E−11	E−10	E−11	E−10	E−11	E−11	E−11	E−11
nos2	–	–	–	–	–	–	–	–	–	–	–	–	–	–
nos3	396	5.71	667	2.51	E−11	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−13	E−11
nos4	204	0.21	202	0.14	E−11	E−11	E−11	E−11	E−11	E−13	E−14	E−14	E−14	E−12
nos5	2232	6.25	2505	4.70	E−11	E−11	E−11	E−11	E−11	E−12	E−12	E−12	E−12	E−11
nos7	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 12
Check of correctness for the rightmost eigenpairs.

√	1138bus	MATLAB	.2105105115E+05	.2194783633E+05	.3000130387E+05	.3001049004E+05	.3014879442E+05
		DAVIDSON	.2105105115E+05	.2194783633E+05	.3000130387E+05	.3001049004E+05	.3014879442E+05
		ARPACK	.2105105115E+05	.2194783633E+05	.3000130387E+05	.3001049004E+05	.3014879442E+05
√	662bus	MATLAB	.1422994613E+04	.1424742094E+04	.1617986726E+04	.2220033359E+04	.4008113925E+04
		DAVIDSON	.1422994613E+04	.1424742094E+04	.1617986726E+04	.2220033359E+04	.4008113925E+04
		ARPACK	.1422994613E+04	.1424742094E+04	.1617986726E+04	.2220033359E+04	.4008113925E+04
√	bcsstk01	MATLAB	.2018372795E+10	.2207957140E+10	.2220593407E+10	.2970424445E+10	.3015179090E+10
		DAVIDSON	.2018372795E+10	.2207957140E+10	.2220593407E+10	.2970424445E+10	.3015179090E+10
		ARPACK	.2018372795E+10	.2207957140E+10	.2220593407E+10	.2970424445E+10	.3015179090E+10
√	bcsstk02	MATLAB	.1438284448E+05	.1511295789E+05	.1621278900E+05	.1665103995E+05	.1822574862E+05
		DAVIDSON	.1438284448E+05	.1511295789E+05	.1621278900E+05	.1665103995E+05	.1822574862E+05
		ARPACK	.1438284448E+05	.1511295789E+05	.1621278900E+05	.1665103995E+05	.1822574862E+05
√	bcsstk05	MATLAB	.5255490753E+07	.5463067787E+07	.5808326738E+07	.5808726611E+07	.6197287056E+07
		DAVIDSON	.5255490753E+07	.5463067787E+07	.5808326738E+07	.5808726611E+07	.6197287056E+07
		ARPACK	.5255490753E+07	.5463067787E+07	.5808326738E+07	.5808726611E+07	.6197287056E+07
√	bcsstk08	MATLAB	.1686207543E+11	.2218776454E+11	.2711507179E+11	.4416405745E+11	.7657033866E+11
		DAVIDSON	.1686207543E+11	.2218776454E+11	.2711507179E+11	.4416405745E+11	.7657033866E+11
		ARPACK	.1686207543E+11	.2218776454E+11	.2711507179E+11	.4416405745E+11	.7657033866E+11
√	bcsstk09	MATLAB	.6654598161E+08	.6679820601E+08	.6720297873E+08	.6720297873E+08	.6760303645E+08
		DAVIDSON	.6654597896E+08	.6679820601E+08	.6720297873E+08	.6720297873E+08	.6760303645E+08
		ARPACK	.6654598161E+08	.6679820601E+08	.6720297873E+08	.6720297873E+08	.6760303645E+08
√	bcsstk10	MATLAB	.4346427154E+08	.4393949521E+08	.4431269100E+08	.4458118573E+08	.4474305592E+08
		DAVIDSON	.4346427154E+08	.4393949521E+08	.4431269100E+08	.4458118573E+08	.4474305592E+08
		ARPACK	.4346427154E+08	.4393949521E+08	.4431269100E+08	.4458118573E+08	.4474305592E+08
√	bcsstk11	MATLAB	.6550590910E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
		DAVIDSON	.6550590910E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
		ARPACK	.6538718159E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
●●	▷ bcsstk12	MATLAB	.6550590910E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
		DAVIDSON	.6550590910E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
	●● ▷	ARPACK	.6538718159E+09	.6550590910E+09	.6550590910E+09	.6556063155E+09	.6556063155E+09
√	bcsstk19	MATLAB	.1911085254E+15	.1916338492E+15	.1916338492E+15	.1922160604E+15	.1922160604E+15
		DAVIDSON	.1911085254E+15	.1916338492E+15	.1916338492E+15	.1922160604E+15	.1922160604E+15
		ARPACK	.1911085254E+15	.1916338492E+15	.1916338492E+15	.1922160604E+15	.1922160604E+15
√	bcsstk27	MATLAB	.3035552754E+07	.3128789997E+07	.3231908139E+07	.3344127105E+07	.3465059451E+07
		DAVIDSON	.3035552754E+07	.3128789997E+07	.3231908139E+07	.3344127105E+07	.3465059451E+07
		ARPACK	.3035552754E+07	.3128789997E+07	.3231908139E+07	.3344127105E+07	.3465059451E+07
√	bcsstm04	MATLAB	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00
		DAVIDSON	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00
		ARPACK	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00	.1728285735E+00
●●	▷ bcsstm06	MATLAB	.7602410190E+04	.7602410228E+04	.7602410228E+04	.7602410228E+04	.7602410228E+04
		DAVIDSON	.7602410190E+04	.7602410228E+04	.7602410228E+04	.7602410228E+04	.7602410228E+04
	●● ▷	ARPACK	.7028218361E+04	.7602410190E+04	.7602410190E+04	.7602410228E+04	.7602410228E+04
●●	▷ bcsstm10	MATLAB	.2813595337E+05	.2828847122E+05	.2843798632E+05	.2868310004E+05	.2883090837E+05
	●● ▷	DAVIDSON	.2809741054E+05	.2828847122E+05	.2843798632E+05	.2868310004E+05	.2883090837E+05
		ARPACK	.2813595337E+05	.2828847122E+05	.2843798632E+05	.2868310004E+05	.2883090837E+05
√	bcsstm12	MATLAB	.1319142097E+02	.1325744318E+02	.1325744318E+02	.1341559058E+02	.1341559058E+02
		DAVIDSON	.1319142097E+02	.1325744318E+02	.1325744318E+02	.1341559058E+02	.1341559058E+02
		ARPACK	.1319142097E+02	.1325744318E+02	.1325744318E+02	.1341559058E+02	.1341559058E+02
√	bcsstm27	MATLAB	.3460249067E+04	.3558168823E+04	.3669122473E+04	.3792212298E+04	.3927163488E+04
		DAVIDSON	.3460249067E+04	.3558168823E+04	.3669122473E+04	.3792212298E+04	.3927163488E+04
		ARPACK	.3460249067E+04	.3558168823E+04	.3669122473E+04	.3792212298E+04	.3927163488E+04
√	gr3030	MATLAB	.1187843564E+02	.1192869592E+02	.1192869592E+02	.1195905988E+02	.1195905988E+02
		DAVIDSON	.1187843564E+02	.1192869592E+02	.1192869592E+02	.1195905988E+02	.1195905988E+02
		ARPACK	.1187843564E+02	.1192869592E+02	.1192869592E+02	.1195905988E+02	.1195905988E+02
√	lunda	MATLAB	.2122131218E+09	.2165941433E+09	.2197883625E+09	.2210402147E+09	.2238540644E+09
		DAVIDSON	.2122131218E+09	.2165941433E+09	.2197883625E+09	.2210402147E+09	.2238540644E+09
		ARPACK	.2122131218E+09	.2165941433E+09	.2197883625E+09	.2210402147E+09	.2238540644E+09
√	nos2	MATLAB	.1571916938E+12	.1572257837E+12	.1572523014E+12	.1572712445E+12	.1572826110E+12
		DAVIDSON	.1571916938E+12	.1572257837E+12	.1572523014E+12	.1572712445E+12	.1572826110E+12
		ARPACK	.1571916938E+12	.1572257837E+12	.1572523014E+12	.1572712445E+12	.1572826110E+12
√	nos3	MATLAB	.6720400662E+03	.6758195632E+03	.6772072368E+03	.6845858636E+03	.6899039606E+03
		DAVIDSON	.6720400662E+03	.6758195632E+03	.6772072368E+03	.6845858636E+03	.6899039606E+03
		ARPACK	.6720400662E+03	.6758195632E+03	.6772072368E+03	.6845858636E+03	.6899039606E+03
√	nos4	MATLAB	.7954146747E+00	.7979687852E+00	.8158840147E+00	.8367012217E+00	.8491377838E+00
		DAVIDSON	.7954146747E+00	.7979687852E+00	.8158840147E+00	.8367012217E+00	.8491377838E+00
		ARPACK	.7954146747E+00	.7979687852E+00	.8158840147E+00	.8367012217E+00	.8491377838E+00
√	nos5	MATLAB	.5090933789E+06	.5279354620E+06	.5364439368E+06	.5612720376E+06	.5820291156E+06
		DAVIDSON	.5090933789E+06	.5279354620E+06	.5364439368E+06	.5612720376E+06	.5820291156E+06
		ARPACK	.5090933789E+06	.5279354620E+06	.5364439368E+06	.5612720376E+06	.5820291156E+06
√	nos7	MATLAB	.6836227287E+07	.8224301537E+07	.8224302770E+07	.8224302770E+07	.9864030300E+07
		DAVIDSON	.6836227287E+07	.8224301537E+07	.8224302770E+07	.8224302770E+07	.9864030300E+07
		ARPACK	.6836227287E+07	.8224301537E+07	.8224302770E+07	.8224302770E+07	.9864030300E+07

Table 13
Check of correctness for the leftmost eigenpairs.

1138bus	MATLAB	.3516860006E−02	.9862234734E−01	.1241279307E+00	.1768149305E+00	.1831768532E+00
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
√662bus	MATLAB	.5047169019E−02	.1623680950E+00	.2638129026E+00	.3351872373E+00	.3658843786E+00
	DAVIDSON	.5047169019E−02	.1623680950E+00	.2638129026E+00	.3351872373E+00	.3658843786E+00
	ARPACK	.5047169019E−02	.1623680950E+00	.2638129026E+00	.3351872373E+00	.3658843786E+00
√bcsstk01	MATLAB	.3417267563E+04	.8970009818E+04	.1083565548E+05	.2232699142E+05	.5163408924E+05
	DAVIDSON	.3417267563E+04	.8970009818E+04	.1083565548E+05	.2232699141E+05	.5163408923E+05
	ARPACK	.3417267563E+04	.8970009819E+04	.1083565549E+05	.2232699142E+05	.5163408923E+05
√bcsstk02	MATLAB	.4214073733E+01	.4300382397E+01	.5258221526E+01	.2636205495E+02	.3805932197E+02
	DAVIDSON	.4214073733E+01	.4300382397E+01	.5258221526E+01	.2636205495E+02	.3805932197E+02
	ARPACK	.4214073733E+01	.4300382397E+01	.5258221526E+01	.2636205495E+02	.3805932197E+02
√bcsstk05	MATLAB	.4339489605E+03	.4430684545E+03	.1442783766E+04	.3226074854E+04	.4151646003E+04
	DAVIDSON	.4339489605E+03	.4430684545E+03	.1442783766E+04	.3226074854E+04	.4151646003E+04
	ARPACK	.4339489605E+03	.4430684545E+03	.1442783766E+04	.3226074854E+04	.4151646003E+04
√bcsstk08	MATLAB	.2946410519E+04	.3494108138E+04	.3539629915E+04	.3643714454E+04	.3805034584E+04
	DAVIDSON	.2946410519E+04	.3494108138E+04	.3539629916E+04	.3643714455E+04	.3805034584E+04
	ARPACK	–	–	–	–	–
√bcsstk09	MATLAB	.7102229057E+04	.2734155790E+05	.2734155790E+05	.5495682031E+05	.7857362575E+05
	DAVIDSON	.7102229057E+04	.2734155790E+05	.2734155790E+05	.5495682031E+05	.7857362575E+05
	ARPACK	.7102229057E+04	.2734155790E+05	.2734155790E+05	.5495682031E+05	.7857362575E+05
bcsstk10	MATLAB	.8535085715E+02	.8548309432E+02	.9500642237E+02	.1036665102E+03	.1167622802E+03
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
bcsstk11	MATLAB	.2964059196E+01	.2965967441E+01	.1076627628E+02	.1098851091E+02	.2039041618E+02
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
bcsstk12	MATLAB	.2964059196E+01	.2965967441E+01	.1076627628E+02	.1098851091E+02	.2039041618E+02
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
bcsstk19	MATLAB	.1434350946E+04	.1612674950E+04	.6469992415E+04	.8181055931E+04	.1296686855E+05
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
√bcsstk27	MATLAB	.1435646443E+03	.1456035900E+03	.1474851421E+03	.1491779411E+03	.1506381265E+03
	DAVIDSON	.1435646443E+03	.1456035900E+03	.1474851421E+03	.1491779411E+03	.1506381265E+03
	ARPACK	–	–	–	–	–
●● ▷ bcsstm04	MATLAB	.0000000000E+00	.0000000000E+00	.0000000000E+00	.0000000000E+00	.0000000000E+00
●● ▷	DAVIDSON	.1974693867E−01	.1974693867E−01	.1974693867E−01	.2317061005E−01	.2317061005E−01
	ARPACK	−.3374579262E−17	−.2492895683E−17	−.9064711524E−18	−.3687665604E−21	.2435499255E−22
●● ▷ bcsstm06	MATLAB	.2199084150E−02	.2199084150E−02	.2199084150E−02	.2199084150E−02	.2199084150E−02
●● ▷	DAVIDSON	.2199084150E−02	.2199084150E−02	.2199084150E−02	.2199084150E−02	.1176958507E+01
	ARPACK	.2199084149E−02	.2199084151E−02	.2199084151E−02	.2199084151E−02	.2199084151E−02
√bcsstm10	MATLAB	−.2060531292E+04	−.2037786261E+04	−.2000370305E+04	−.1949035405E+04	−.1884824806E+04
	DAVIDSON	−.2060531292E+04	−.2037786261E+04	−.2000370305E+04	−.1949035405E+04	−.1884824806E+04
	ARPACK	−.2060531292E+04	−.2037786261E+04	−.2000370305E+04	−.1949035405E+04	−.1884824806E+04
bcsstm12	MATLAB	.2118712112E−04	.2118712115E−04	.2136055145E−04	.2136055147E−04	.2136090589E−04
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
√bcsstm27	MATLAB	−.5958597977E+01	−.5958597963E+01	−.5198055485E+01	−.5198055064E+01	−.4488746706E+01
	DAVIDSON	−.5958597977E+01	−.5958597963E+01	−.5198055485E+01	−.5198055064E+01	−.4488746706E+01
	ARPACK	−.5958597977E+01	−.5958597963E+01	−.5198055485E+01	−.5198055064E+01	−.4488746706E+01
√gr3030	MATLAB	.6146282393E−01	.1531843111E+00	.1531843111E+00	.2439646117E+00	.3050073347E+00
	DAVIDSON	.6146282393E−01	.1531843111E+00	.1531843111E+00	.2439646117E+00	.3050073347E+00
	ARPACK	.6146282393E−01	.1531843111E+00	.1531843111E+00	.2439646117E+00	.3050073347E+00
●● ▷ lunda	MATLAB	.8003510930E+02	.1976505467E+04	.1996764780E+04	.6354111204E+04	.1283833070E+05
●● ▷	DAVIDSON	.3451911578E+08	.3452172302E+08	.4513157486E+08	.4604036289E+08	.4769761385E+08
	ARPACK	.8003510935E+02	.1976505467E+04	.1996764780E+04	.6354111204E+04	.1283833070E+05
nos2	MATLAB	.3084226603E+02	.1233660915E+03	.1392471275E+03	.2775625587E+03	.4934168059E+03
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–
√nos3	MATLAB	.1828839439E−01	.2488598861E+00	.2671801822E+00	.1166601955E+01	.1939512306E+01
	DAVIDSON	.1828839439E−01	.2488598861E+00	.2671801822E+00	.1166601955E+01	.1939512306E+01
	ARPACK	.1828839439E−01	.2488598861E+00	.2671801822E+00	.1166601955E+01	.1939512306E+01
√nos4	MATLAB	.5379528369E−03	.3860235311E−02	.4546036376E−02	.1012314149E−01	.1417578665E−01
	DAVIDSON	.5379528369E−03	.3860235311E−02	.4546036376E−02	.1012314149E−01	.1417578665E−01
	ARPACK	.5379528369E−03	.3860235311E−02	.4546036376E−02	.1012314149E−01	.1417578665E−01
√nos5	MATLAB	.5289948260E+02	.6754297175E+02	.1155911521E+03	.1315636348E+03	.1857169377E+03
	DAVIDSON	.5289948260E+02	.6754297175E+02	.1155911521E+03	.1315636348E+03	.1857169377E+03
	ARPACK	.5289948260E+02	.6754297175E+02	.1155911521E+03	.1315636348E+03	.1857169377E+03
nos7	MATLAB	.4154132324E−02	.1033441946E−01	.1197176140E−01	.1342134128E−01	.1504096094E−01
	DAVIDSON	–	–	–	–	–
	ARPACK	–	–	–	–	–

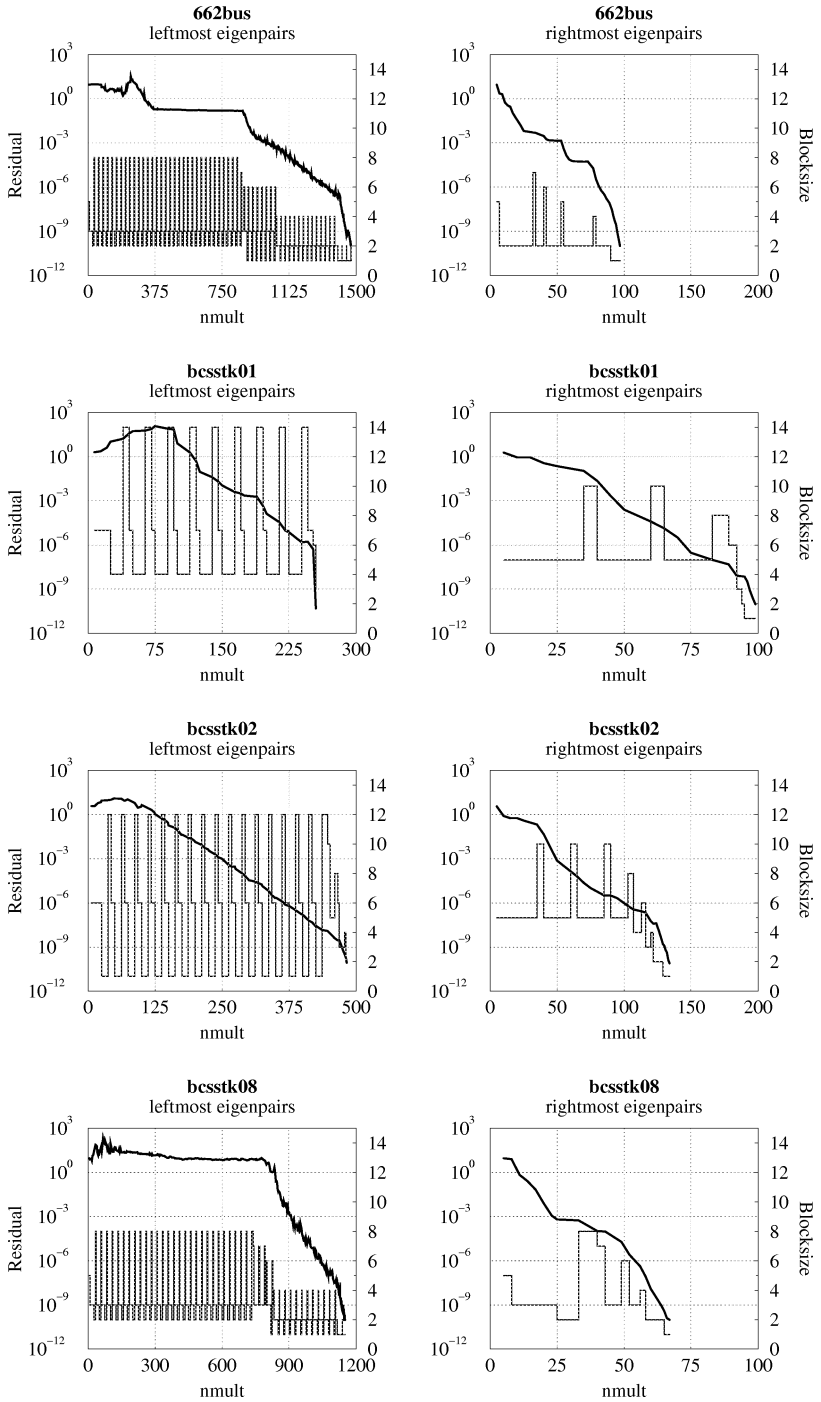

Figure 2. History of convergence of DAVIDSON for some representative best runs.

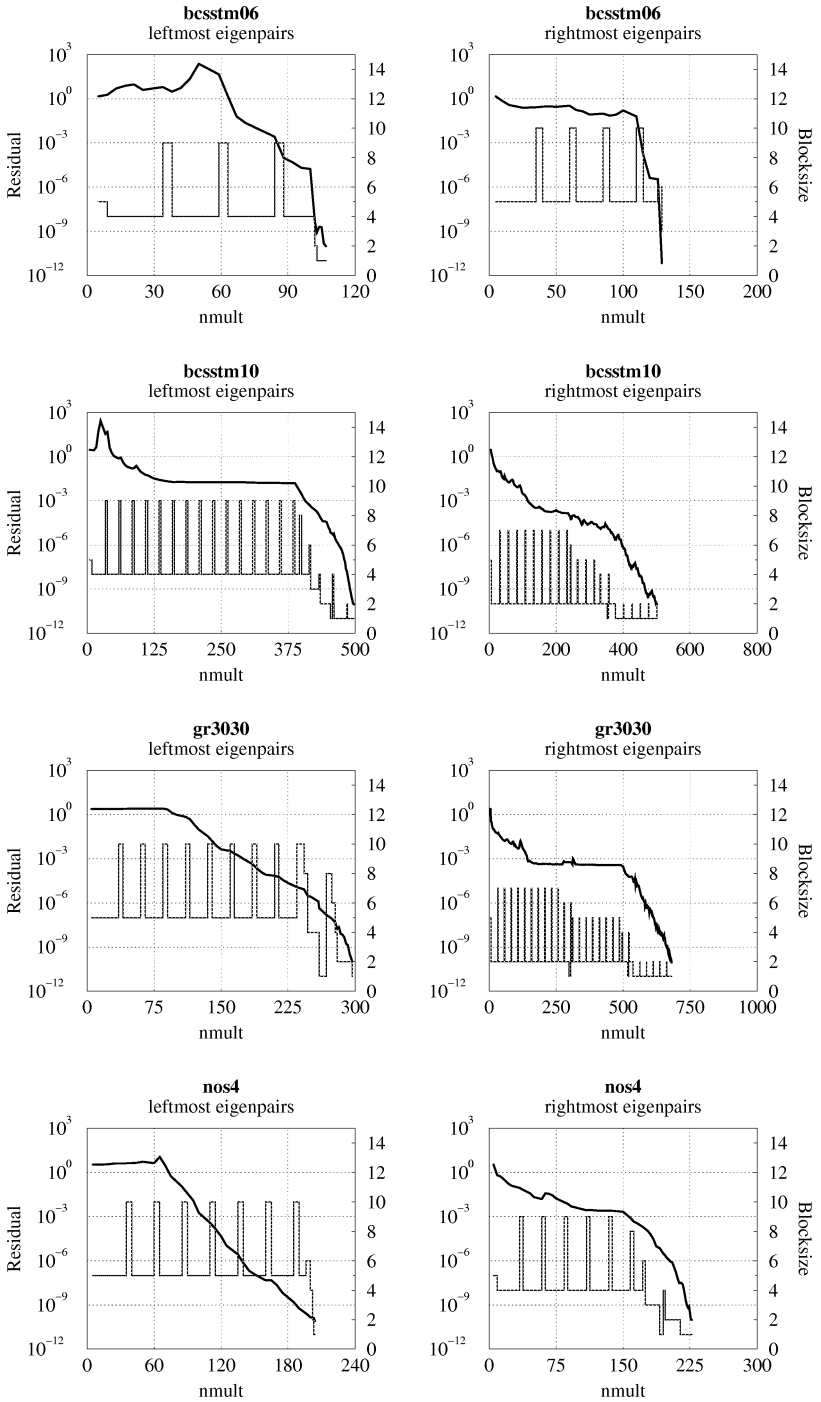

Figure 3. History of convergence of DAVIDSON for some representative best runs.

Table 14
Best timing with tolerance $tol = 10^{-10}$ and basis of size $m = 30$.

	Five rightmost eigenpairs						Five leftmost eigenpairs					
	ARPACK		DAVIDSON				ARPACK		DAVIDSON			
	nmult	time	nmult	time	block	corr	nmult	time	nmult	time	block	corr
1138bus	78	0.27	125	0.41	3	NC	–	–	$nconv = 4$		1	IC
662bus	57	0.12	103	0.21	3	NC	2716	6.08	1447	5.72	4	IC
bcsstk01	35	0.03	87	0.07	5	GS	2590	1.70	224	0.21	7	GS
bcsstk02	58	0.06	116	0.14	5	GS	297	0.28	453	0.53	6	GS
bcsstk05	60	0.07	141	0.21	3	NC	755	0.79	606	1.38	2	GS
bcsstk08	35	0.12	58	0.20	2	NC	3746	15.21	1136	5.89	4	GS
bcsstk09	243	1.06	782	3.78	6	GS	917	3.98	$nconv = 4$		2	GS
bcsstk10	108	0.48	223	1.12	2	NC	–	–	–	–	–	–
bcsstk11	242	1.50	207	2.24	1	DC	–	–	–	–	–	–
bcsstk12	242	1.50	207	2.24	1	DC	–	–	–	–	–	–
bcsstk19	104	0.30	133	0.46	2	NC	–	–	–	–	–	–
bcsstk27	82	0.53	181	1.25	2	NC	–	–	4245	71.52	1	GS
bcsstm04	97	0.09	60	0.06	5	IC	58	0.05	–	–	–	–
bcsstm06	130	0.19	124	0.19	5	IC	1656	2.31	90	0.15	5	IC
bcsstm10	177	0.81	451	2.26	2	NC	277	1.25	409	2.47	4	GS
bcsstm12	130	0.69	190	1.68	2	GS	–	–	–	–	–	–
bcsstm27	83	0.54	178	1.23	2	NC	2268	15.26	3648	29.75	6	GS
gr3030	334	1.04	719	3.13	3	GS	259	0.80	296	1.79	5	IC
lunda	105	0.11	225	0.29	5	GS	2704	2.75	186	0.24	5	GS
nos2	950	3.02	300	1.25	5	IC	–	–	–	–	–	–
nos3	133	0.51	306	1.35	2	NC	713	2.79	428	5.40	5	IC
nos4	106	0.09	227	0.24	5	GS	180	0.14	184	0.22	5	IC
nos5	104	0.21	191	0.42	5	GS	2211	4.37	1032	4.41	7	IC
nos7	57	0.14	113	0.32	2	NC	–	–	–	–	–	–

on matrices stored under the Harwell–Boeing storage format, Compressed Column Storage format and the Coordinates storage format. Included in the work were wide-ranging numerical experiments performed on the Power Challenge in serial mode and they have contributed in illustrating the usefulness of the package and in showing that with our non-problem specific choice of preconditioner and tuning of parameters, the code compares favorably with the implicitly restarted Arnoldi algorithm ARPACK in speed as well as in scope. The package is written in a friendly and modular style allowing for subsequent additions and modifications by the user. At the core of the package is the variable block Davidson routine itself, which is "matrix-free" and incorporates a deflation technique. Moreover, this core routine is ready to be blended with user-supplied preconditioners in a way similar to those provided in this suite (diagonal, Gauss–Seidel, Incomplete Choleski, Exponential). Users are encouraged to adapt their own specialized preconditioners based on the knowledge of the specificities of their matrices.

Table 15
Best timing with tolerance $tol = 10^{-10}$ and basis of size $m = 40$.

	Five rightmost eigenpairs						Five leftmost eigenpairs					
	ARPACK		DAVIDSON				ARPACK		DAVIDSON			
	nmult	time	nmult	time	block	corr	nmult	time	nmult	time	block	corr
1138bus	76	0.27	46	0.05	5	NC	–	–	4023	21.82	6	IC
662bus	45	0.10	97	0.14	3	NC	1872	4.64	958	4.89	3	IC
bcsstk01	45	0.03	199	0.25	5	GS	1269	1.23	180	0.18	7	GS
bcsstk02	45	0.04	60	0.07	5	IC	248	0.26	346	0.51	6	GS
bcsstk05	79	0.09	204	0.32	5	GS	668	0.79	583	1.33	3	GS
bcsstk08	45	0.17	140	0.23	4	GS	5008	22.04	992	6.25	3	GS
bcsstk09	244	1.12	65	0.20	2	GS	951	4.42	3729	21.30	6	GS
bcsstk10	113	0.53	157	0.42	4	GS	–	–	$nconv = 1$		1	GS
bcsstk11	245	1.59	669	3.28	3	GS	–	–	–	–	–	–
bcsstk12	245	1.59	409	2.36	2	NC	–	–	–	–	–	–
bcsstk19	107	0.34	92	0.21	3	NC	–	–	–	–	–	–
bcsstk27	80	0.52	58	0.20	2	NC	–	–	3622	67.40	1	GS
bcsstm04	76	0.07	629	3.44	6	GS	167	0.15	2765	999.00	1	IC
bcsstm06	207	0.33	200	1.12	2	NC	1543	2.44	99	0.25	3	IC
bcsstm10	179	0.86	192	2.33	1	DC	283	1.36	366	2.56	3	GS
bcsstm12	144	0.81	192	2.31	1	DC	–	–	–	–	–	–
bcsstm27	111	0.75	246	1.61	2	NC	1900	13.42	2744	24.92	6	GS
gr3030	341	1.15	146	0.46	3	NC	211	0.69	270	1.73	5	IC
lunda	112	0.13	164	1.30	2	NC	1987	2.29	175	0.26	5	GS
nos2	773	2.65	174	1.29	2	NC	–	–	–	–	–	–
nos3	148	0.60	272	1.30	5	IC	647	2.71	350	4.67	5	IC
nos4	112	0.10	288	1.55	2	NC	182	0.16	157	0.22	5	IC
nos5	110	0.23	110	0.29	4	NC	1909	4.16	837	3.65	7	IC
nos7	76	0.20	115	0.41	3	NC	–	–	–	–	–	–

References

[1] E. Anderson et al., *LAPACK User's Guide* (SIAM, Philadelphia, PA, 1992).

[2] G. Cisneros and C.F. Bunge, An improved computer program for eigenvectors and eigenvalues of large configuration interaction matrices using the algorithm of Davidson, Comput. Chemistry 8(8) (1986) 157–160.

[3] M. Crouzeix, B. Philippe and M. Sadkane, The Davidson method, SIAM J. Sci. Comput. 15(1) (1994) 62–76.

[4] J.W. Daniel, W.B. Gragg, L. Kaufman and G.W. Stewart, Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization, Math. Comp. 30(136) (1976) 772–795.

[5] E.R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, Comput. Phys. 17 (1975) 87–94.

[6] I.S. Duff, R.G. Grimes and J.G. Lewis, Sparse matrix test problems, ACM Trans. Math. Software 15 (1989) 1–4.

[7] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd ed. (Johns Hopkins University Press, Baltimore, MD, 1989).

[8] A. Jennings and W.J. Stewart, A simultaneous iteration algorithm for real matrices, ACM Trans. Math. Software 7(2) (1981) 184–198.

[9] M.T. Jones and P.E. Plassmann, An improved incomplete Cholesky factorization, ACM Trans. Math. Software 21(1) (1995) 6–17.
[10] M.T. Jones and P.E. Plassmann, Algorithm 740: Fortran subroutines to compute improved incomplete Cholesky factorizations, ACM Trans. Math. Software 21(1) (1995) 18–19.
[11] N. Kosugi, Modification of the Liu–Davidson method for obtaining one or simultaneously several eigensolutions of a large real-symmetric matrix, Comput. Phys. 55 (1984) 426–436.
[12] R.B. Lehoucq, Analysis and implementation of an implicitly restarted Arnoldi iteration, Ph.D. thesis, Department of Computational and Applied Mathematics, Rice University (1995).
[13] R.B. Lehoucq and J.A. Scott, An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices, Technical Report MCS-P547-1195, Argonne National Lab. (1996).
[14] R.B. Lehoucq, D.C. Sorensen and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi methods* (SIAM, Philadephia, PA, 1998).
[15] B. Liu, The simultaneous expansion for the solution of several of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, in: *Numerical Algorithms in Chemistry: Algebraic Method*, eds. C. Moler and I. Shavitt (LBL-8158 Lawrence Berkeley Lab., 1978) pp. 49–53.
[16] K. Meerbergen and M. Sadkane, Using Krylov approximations to the matrix exponential operator in Davidson's method, to appear in Appl. Numer. Math.
[17] R.B. Morgan and D.S. Scott, Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices, SIAM J. Sci. Statist. Comput. 7(3) (1986) 817–825.
[18] Y. Saad, SPARSKIT: a basic tool kit for sparse matrix computation, version 2, Technical Report, Computer Science Department, University of Minnesota, Minneapolis, MN.
[19] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Algorithms and Architectures for Advanced Scientific Computing (Manchester University Press, Manchester, UK, 1992).
[20] M. Sadkane, Block Arnoldi and Davidson methods for unsymmetric large eigenvalue problems, Numer. Math. 64 (1993) 195–211.
[21] R.B. Sidje, EXPOKIT: Software package for computing matrix exponentials, ACM Trans. Math. Software 24(1) (1998) 130–156.
[22] G.L.G. Sleijpen and H.A. Van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425.
[23] D.C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, SIAM J. Matrix Anal. Appl. 13 (1992) 357–385.
[24] A. Stathopoulos and C.F. Fischer, A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix, Comput. Phys. Comm. 79(2) (1994) 268–290.
[25] J. Weber, R. Lacroix and G. Wanner, The eigenvalue problem in configuration interaction calculations: a computer program based on a new derivation of the algorithm of Davidson, Comput. Chemistry 4(2) (1980) 55–60.