A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix

Andreas Stathopoulos, Charlotte F. Fischer
Computer Science Department, Vanderbilt University,
Nashville, TN 37235, USA

October 31, 1993

Abstract

A program is presented for determining a few selected eigenvalues and their eigenvectors on either end of the spectrum of a large, real, symmetric matrix. Based on the Davidson method, which is extensively used in quantum chemistry/physics, the current implementation improves the power of the original algorithm by adopting several extensions. The matrix-vector multiplication routine that it requires is to be provided by the user. Different matrix formats and optimizations are thus feasible. Examples of an efficient sparse matrix representation and a matrix-vector multiplication are given. Some comparisons with the Lanczos method demonstrate the efficiency of the program.

PROGRAM SUMMARY

Title of program: DVDSON
Catalogue Number: To be assigned
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue).
Licensing provisions: none
Computer: Sun-3/80, Sun SPARCstation IPC, Intel iPSC/860.
Operating system: SunOS Release 4.1.2, UNIX System V/386 Release 3.2.
Programming language used: FORTRAN77
Peripherals used: None by DVDSON. The user supplied routine for matrix-vector multiplication may make use of external storage.
No. of lines in distributed program and test data: DVDSON: 1036 lines, sample driver, matrix-vector multiplication, gather, scatter and dinit: 364 lines, sample output: 120 lines.
Library routines used: LAPACK, BLAS.
Keywords: Eigenvalue, eigenvector, sparse, symmetric matrix, matrix-vector multiplication, Davidson method, Lanczos method, electronic structure, computational physics, computational quantum chemistry.
Nature of physical problem
Finding a few extreme eigenpairs of a real, symmetric matrix is of great importance in scientific computations. Examples abound in structural engineering, quantum chemistry and electronic structure physics [1,2]. The matrices involved are usually too large to be efficiently solved using standard methods. Moreover, their large size often prohibits full storage forcing various sparse representations. Even sparse representations cannot always be stored in main memory [3]. Thus, an iterative method is needed that converges rapidly to the few extreme eigenpairs required and takes advantage of the symmetry, sparsity and possibly the secondary storage of the matrix.
Method of Solution
This program implements a version of the original Davidson method [4], focusing on high performance on vector processors and adopting many previously proposed extensions [3,5,6] which have not appeared together in a published program. The features of the current version are:

- Selected eigenpairs at either the lower or the higher end of the spectrum can be found, without enforcing convergence for interspersed unwanted ones.

- The basis of eigenvectors as well as all other program arrays are kept in memory.
- The matrix-vector multiplication routine must be supplied by the user.
- A block method is used, where more than one vector may be targeted in each iteration.
- The vectors offering optimal improvement are targeted without the need to explicitly compute all residuals.
- A reorthogonalization procedure is included for numerical stability.
- Initial eigenvector estimates can either be provided by the user or generated by the program.
- The user can control basis size, block size, reorthogonalization, and convergence through three criteria.
- All floating-point intensive code is vectorizable.

Restrictions on the complexity of the problem
For performance improvements, DVDSON keeps the basis in memory which may cause memory overflow problems if exceedingly large matrices are used. This program bears two of the Davidson method's weaknesses: convergence is slow when the diagonal of the matrix is constant and not guaranteed for the required eigenpair when the matrix is permutationally equivalent to a block diagonal matrix.
Typical running time
For each eigenpair, the Davidson method takes 10-30 iterations if the block size is one. For larger block sizes the number of iterations decreases significantly.
References

1 B.N. Parlett, SIAM J. Sci. Stat. Comput. 5 (1984) 590.

2 C.F. Fischer, Comput. Phys. Commun. 64 (1991) 369.

3 E.R. Davidson, Comput. Phys. Commun. 53 (1989) 49.

4 E.R. Davidson, J. Comput. Phys. 17 (1975) 87.

5 G. Cisneros, M. Berrondo and C.F. Bunge, Compu. Chem. 10 (1986) 281.

6 J. Weber, R. Lacroix and G. Wanner, Comput. Chem. 4 (1980) 55.

LONG WRITEUP

1 Introduction

Many scientific applications require the solution of the eigenvalue problem,

$$Ax_i = \lambda_i x_i, \tag{1}$$

where λ_i are a few of the lowest or highest (extreme) eigenvalues of the matrix A, and x_i their corresponding eigenvectors (together they form eigenpairs). In these applications it is common for A to be real, symmetric, and frequently very large and sparse. Examples can be found both in engineering and in science. In structural engineering the stiffness matrix obtained by the analysis of small vibrations of structures is usually of order $O(10^4)$ or larger and only a few lowest eigenpairs are required [1, 2]. In electronic structure physics, the order of Hamiltonian matrix obtained by Configuration Interaction (CI) methods can differ by several orders of magnitude for different models. In quantum chemistry, matrices of order $O(10^6)$ are no longer unusual. On serial computers, the matrix elements can only be stored on disk despite the sparsity of these matrices. For the largest of these matrices, the disk capacity of most supercomputers is exceeded. Problems like the latter can only be overcome if the matrix elements can be regenerated as needed [3, 4]. In atomic physics applications [5, 6, 7], problems are several orders of magnitude smaller ($O(10^3) - O(10^5)$) and the smallest of these may be stored in sparse format in memory. A few extreme eigenpairs are required, although occasionally an interior eigenpair corresponding to an excited state is of interest.

The above examples share the following common characteristics: the matrices involved cannot be stored in memory or on disk in full; their sparse nature calls for special data structures; and only few extreme eigenpairs are needed. For small orders the problem can be treated with standard routines from packages like EISPACK, HARWELL, etc., [8, 9]. However, in problems such as the above ones standard methods cannot be applied or even if they could, they would be extremely inefficient since the whole spectrum is found or the matrices are transformed to tridiagonal form which has proved inefficient [10]. In atomic physics and quantum chemistry good initial estimates to the eigenpairs desired can often be provided that standard methods do not exploit. Finally, since matrix generation is usually expensive, it is common that the nonzero elements are stored as they are calculated in an unordered form [3]. In this case an algorithm that performs advanced matrix operations or modifies the matrix, would be infeasible. Therefore a rapidly convergent iterative method is needed that manages sparse structures and relies only on matrix-vector multiplication.

Many algorithms and programs have appeared during the last twenty years. E. R. Davidson developed a method [11] that uses perturbation theory to take advantage of the sparsity and the large diagonal-dominance ratio of the matrices appearing in quantum chemistry. The term diagonal-dominance ratio is used in this paper to refer to the ratio $d = \min_{i,j}|(A_{ii} - A_{jj})/A_{ij}|$. Similar properties are shared by matrices that appear in other electronic structure computations, such as atomic physics [10]. A general purpose Davidson program that takes advantage of the above special properties but does not depend on the specific representations, would expedite computations in these fields as well.

2 The Davidson Method

Among several methods proposed in the literature, the Lanczos method is the most widely used one. The Lanczos iteration builds an orthogonal basis for the Krylov subspace,

$$\mathcal{K}(A, g, P) = \text{span}\{g, Ag, \ldots, A^P g\}, \tag{2}$$

from which the required eigenvectors are approximated through a Rayleigh-Ritz procedure [12, 13]. The power of the method lies in the fact that the orthogonal basis is built through an easy-to-compute three-term recurrence and the projection of A onto $\mathcal{K}(A, g, P)$ is a tridiagonal matrix of order P. The main concern with this method is that a large number of iterations may be required before a sufficiently invariant subspace is found.

The Davidson method reduces the number of iterations at the expense of a more complicated step. If x_i is the current approximation of the i_{th} eigenvector, from perturbation theory, an improvement on the Lanczos correction vector to be added to the basis is given by:

$$b = (\rho I - M)^{-1} Res(x_i), \tag{3}$$

where $\rho = x_i^T A x_i / x_i^T x_i$ is the Rayleigh quotient, $Res(x_i) = (A - \rho I)x_i$ the residual of x_i, and $(\rho I - M)$ an easily inverted approximation to $(\rho I - A)$[3, 14]. If (ρ, x_i) is close to an eigenpair, equation (3) provides second order convergence in ρ. The vectors built in this way comprise the basis $B = \{b_1, \ldots, b_P\}$ of a subspace that provides better information about the required eigenvectors than the $\mathcal{K}(A, g, P)$ subspace does. Davidson used this perturbation scheme to efficiently compute estimates to the lowest energy levels and the corresponding wave functions of the Schrödinger operator [11]. In CI calculations matrices usually have a large diagonal-dominance ratio and therefore the matrix of diagonals of A can be considered a good approximation to A. Davidson chooses $M = \text{diag}(A)$ and thus the new vector to be included in the basis in the j_{th} step is

$$b^{(j)} = (\rho I - \text{diag}(A))^{-1}(A - \rho I)x_i^{(j-1)}. \tag{4}$$

In order to avoid division by zero when the Rayleigh quotient, ρ, happens to be equal to some diagonal element, an arbitrary small value can be used instead. $x_i^{(j-1)}$ is the approximation of the i_{th} eigenvector on the $(j-1)_{th}$ step. The way that this is chosen is discussed later.

Assuming that the K lowest eigenpairs are required, a brief description of the algorithm as proposed initially follows:
Step 0: Set $P = K$. Compute the initial Basis $B = \{b_1, \ldots, b_P\} \in \Re^{N\times P}$, $D = AB = \{d_1, \ldots, d_P\}$, and the projection $S = B^T AB = B^T D$.
Repeat until converged steps 1 through 8:

1. Solve the symmetric eigenvalue problem: $SC = CU$ (size $P \times P$).
2. Target one of the K sought eigenpairs, say (u, c).
3. If the basis size is maximum restart: $D \leftarrow DC,\ B \leftarrow BC,\ C = I_K,\ S = U,\ P = K$.
4. Compute $R = (diag(A) - uI)^{-1}(Dc - uBc)$.
5. Orthogonalize: $b_{new} = R - \sum b_i b_i^T R$, normalize: $b_{new} \leftarrow b_{new}/\|b_{new}\|$.
6. Matrix-vector multiplication: $d_{new} = Ab_{new}$
7. Include b_{new} in B and d_{new} in D. Increase P.
8. Compute the new column of S: $S_{i,P} = b_i^T d_P,\ i = 1, \ldots, P$.

In step 0, if no initial estimates are available, the method picks the K unit vectors corresponding to the K lowest diagonal elements. It is then possible to construct the matrix $D^{(0)} = AB^{(0)}$ without matrix-vector multiplication.

Comments on the method

The diagonal operator of equation (4), used for improved correction vectors, also provides additional flexibility to the method. In each step a correction to a particular eigenvector is found and this targeted eigenpair is improved. In contrast, the Lanczos method builds a Krylov space of which each new vector incorporates information for the whole space. Thus improvement occurs both to wanted and to some redundant eigenvectors. An immediate but very important extension to the above flexibility is the capability of the method to converge directly to higher eigenpairs without the need to accurately compute all lower ones. In a general sense, the diagonal operator is similar to the inverse iteration operator since both can be used for finding eigenvalues close to ρ. Since ρ is the Rayleigh quotient that converges to the eigenvalue, selective convergence can be achieved.

The power of the method is not restricted to the use of the diagonal operator. The version of subspace iteration adopted, where the basis is redefined when it becomes reasonably large, allows for a large number of iterations without the need to orthonormalize a very large basis. Since explicit orthogonalization is used, this saves both time and round-off errors per step. It also reduces dramatically the storage requirements of the method. Finally, the use of a matrix-vector operator as the only means of accessing the matrix A, encapsulates the difficulties of coping with the size and format of the matrix.

3 Implementation Characteristics

Many extensions have been suggested to improve the method's speed of convergence, accuracy and functionality. Most of those reported to give favorable results are implemented in the current program. Even though all of the implemented extensions have been tested, they have never appeared in a single published program. The following discusses the characteristics of each extension and the form it is adopted.

Diagonal Operator
The obvious suggestion is to use a tridiagonal operator instead of the diagonal one [14]. Crouzeix et

al. have reported cases with significant improvements over the original method [15]. However, they also report that this may not always be the case. The factorization of the tridiagonal matrix requires additional storage either in memory or on disk. In case of disk storage there is considerable overhead involved in each iteration. Moreover, since the matrices in electronic structure calculations have a very large diagonal-dominance ratio, the convergence improvement using tridiagonal operator, would probably not justify the additional overheads. The original Davidson operator was retained in the current version.

Targeting Vectors

The next important issue concerns the choice of the eigenvectors to be targeted for improvement in the next iteration. When eigenpairs with indices $i_1, i_2, \ldots, i_K$ are wanted, the optimal choice is to compute all residuals Res_{i_j}, and choose i to correspond to the maximum $\|\mathrm{Res}_i\|$. However, this could be quite an expensive procedure. Davidson has proposed several alternative ways to choose the targeted x_i to be used in equation (8) [16].

(1) Cycle i through the indices $i_1, i_2, \ldots, i_K$ of unconverged eigenpairs.
(2) Choose i from the indices of unconverged eigenpairs, with the largest coefficient $|C_{Pi}|$ of the last basis vector that entered B.
(3) Start i at i_1 and increment to the next index when the x_{i_1} converges.
(4) Choose i to correspond to an eigenvector with a desired pattern of coefficients.

As he mentions, choice (2) leads to the most uniform convergence for all x_{i_j} [17]. Choice (2) seems to imply that $|C_{Pi}| > |C_{Pj}| \Rightarrow \|\mathrm{Res}_i\| > \|\mathrm{Res}_j\|$. In general, this is not true but it is a good approximation that provides the uniform convergence. However, in order to prevent a halt in the augmentation of the basis, whenever a targeted vector converges, another vector must be targeted in the same iteration. Hence, more than one largest coefficients should be given by choice (2). The current program employs this technique. In each step all the coefficients $|C_{Pi}|$ of the unconverged eigenvectors are sorted and the first *block-size* ones (see next paragraph) are targeted. If some of these vectors converge, the next coefficients in order are chosen. Once an eigenvector has converged it is never targeted again.

The root-homing method of Butscher et al. [18], used in choice (4), is a useful but specialized extension that is not implemented in the present program.

Block Method

A very popular extension to many iterative methods is the block-method. Liu [19] has suggested a block-Davidson method where all required and non converged eigenvectors are targeted in the same iteration. Thus several new basis vectors are generated and the memory requirements of the method increase. This methodology is known to improve convergence and to provide a stable remedy for degenerate eigenvalues. The time benefits of this extension are not immediately obvious. The vectors included in the basis no longer correspond to the best contribution of the one largest $|C_{Pi}|$; thus the basis contains some vectors that do not offer as good a correction. Consequently, more matrix-vector multiplies are performed but the number of iterations is reduced. The power of the method emerges when the access to the matrix is very expensive (matrix stored on disk or recomputed each time). In each iteration the matrix is referenced once and several matrix-vector multiplies are performed. Since the bottleneck of the method for large matrices is the matrix access, reduction in the number of iterations translates to savings in time. However, if the matrix is easily accessible and the bottleneck is the calculation of the matrix-vector multiplication, this method takes more time than the original one. Another possible advantage of blocks is the introduction of independent tasks that can be distributed in parallel machines like Cray, Alliant, etc., with relatively little effort. The current program is a block-Davidson method where the size of the block can be adjusted by the user according to the specific problem to finely tune performance. The choice of i is the one described in the previous paragraph.

Reorthogonalization
One of the important requirements of the Davidson method is that the basis be orthonormal. It is possible that orthogonality can be lost for some ill-conditioned problems. Weber et al. [20], in the first published Davidson program, used a reorthogonalization process of the vector entering the basis in case of non-exact orthogonality. This scheme is adopted in this program as well. A threshold is specified above which loss of orthogonality is assumed. When tight residual convergence is required it is advisable that the orthogonality threshold is close to the residual convergence threshold.

Memory vs. Disk
When first introduced, the Davidson method had the big advantage of requiring storage for only two vectors of size N in memory at any time. The reason was that computer memories were small and very expensive while the problems dealt with by computational chemists were large. Thus not only the matrix A was kept on disk but the basis B as well. That introduced significant overhead in each iteration and huge bills for computer charges [1]. Three observations evince the migration of the basis B and sometimes even the matrix A, from disk to memory. First, there are applications that could benefit from the Davidson's speed of convergence but deal only with much smaller problem sizes than those in quantum chemistry. Second, memory size has increased dramatically in recent years. Problems that could not even fit on the disk of most machines 18 years ago, can now fit in the memory of a single supercomputer. Third, the direction of the future in scientific computing passes towards the massively parallel machines [21]. The global memory available on these machines would cater to the needs of most applications. Moreover, highly optimized I/O in these machines is still a research topic. Based on these considerations the current program stores all auxiliary arrays (mainly B and D) in memory while the storage scheme of the matrix A is left user defined. In this way the algorithm is sped up tremendously over previous implementations [20, 22, 23]. In addition, the sizes solvable are not restrictive. On a 64 Mb RAM Sparcstation-2, where the matrix is kept on disk, the lowest eigenpair of matrices of size up to 800,000 can be obtained (see also section (5))!

Size of the Basis
Allowing the basis to grow to a large size reduces the number of iterations since more vectors contribute to the approximation of eigenvectors. When restarting some of this information is compressed thus having less effect. However, a large basis can increase the space and time complexity of each step and it can cause loss of orthogonality [3, 15]. Davidson proposed expanding the basis up to a maximum value and then restarting while Crouzeix et al. [15] suggested testing dynamically the efficiency of the vectors included. If this decreases significantly, the basis is contracted. This strategy is most promising but since experimentation with efficiency thresholds for a wider class of matrices is yet to be done, it has not been implemented in this version of the program.

Matrix Vector Multiply
In block methods the matrix-vector multiplication is actually a matrix-matrix multiplication, since the second matrix is a block of a few vectors. The Davidson method accesses the matrix only through this operator, that should be provided by the user through an external routine. In this way, various sparsity formats and storage schemes can be handled by the same code, and the matrix-vector operation can be optimized to the local architecture [1]. A demonstration of the power of this technique would be to provide this serial algorithm with a parallel matrix-vector

multiplication that operates on a distributed matrix on a parallel computer. Since the time critical step is executed in parallel, the whole algorithm would gain a very good speedup.

Accuracy
The current program improves the accuracy provided by previous Davidson programs [20, 22, 23] since it incorporates both the reorthogonalization procedure and the block method. Convergence close to machine precision was achieved for several test problems. The residual norm is a useful convergence criterion but it does not measure the relative error. Because of $\|Res(x_i^{(j)})\| \simeq \|A-\lambda_i\| \, \|x_i^{(j)} - x_i\|$, [11], the norm of the residual depends on the norm $\|A-\lambda_i\|$. Therefore, if $\|A-\lambda_i\|$ is large, large residual norms should be expected. Scaling of the matrix can be used to alleviate the effects of large norms. Scaling and shifting of the matrix are widely used techniques known to improve numerical stability. These techniques can be easily embedded in the user provided matrix-vector multiplication routine. A more unbiased measure of the eigenvector convergence is the test on $\|x_i^{(j)} - x_i\|$ or more practically on $\|x_i^{(j)} - x_i^{(j-1)}\|$. Since the basis vectors are normalized, the coefficient $C_{P,i}$ of the last vector that entered B is a good estimate of the largest change in $x_i^{(j-1)}$. Convergence of $C_{P,i} < \delta$ yields eigenvector components with accuracy similar to $\log_{10}\delta$ digits [22]. This test was proposed by Davidson and it is adopted in this program as well. When only eigenvalues are of interest, a separate eigenvalue convergence criterion facilitates an earlier but sufficiently accurate termination.

Finally, some characteristics of the program that improve its flexibility and robustness are: the convergence to user specified eigenpairs, finding eigenpairs in the highest end of the spectrum by recording the effect of the negative of the matrix, using initial estimates when available, providing the user with control switches and a detailed error checking and reporting.

The algorithm

The algorithm followed by the program is described next. The notation of the original algorithm is kept. Assume K of the lowest eigenpairs are required with indices $i_1 \leq i_2 \leq \cdots \leq i_K$. Let NUME= i_K. In the case of highest eigenpairs NUME= $N - i_1 + 1$. Let also MBL be the maximum block-size allowed, and LIM be the upper limit on the expanding basis size (LIM = NUME + constant). CRITE, CRITC and CRITR denote the thresholds under which eigenvalues, coefficients and residuals respectively are considered to have converged.

0. If at least NUME initial estimates are not in B, pick unit vectors as described in section (2). Set $D^{(0)} = AB^{(0)}$, $P =$NUME, $P_a = P$. Generate $S^{(0)} = B^{(0)T}AB^{(0)}$.

FOR j=1, maximum number of iterations DO steps 1 to 7.

1. Solve the small problem $S^{(j)}C^{(j)} = C^{(j)}U^{(j)}$, for NUME lowest eigenpairs.
2. If $|U^{(j)}_{i_h i_h} - U^{(j-1)}_{i_h i_h}| <$ CRITE, $h = 1, \ldots, K$, stop.
3. If P_a is the number of vectors added to the basis in the previous iteration, let
   $\mathrm{RL}_h = \max(|C^{(j)}_{p,i_h}|, p = P - P_a + 1, \ldots, P)$, for all x_{i_h} with unconverged residuals.
   Let also P_u be the number of these x_{i_h} for which is satisfied $\mathrm{RL}_h \geq$ CRITC.
   If $P_u = 0$, the coefficients have converged, stop. Otherwise sort $(\mathrm{RL}_h, h = 1, \ldots, P_u)$ in decreasing order of magnitude. The first MBL of them will be targeted.
4. If $(P \geq$ LIM) restart: $B^{(j)} \leftarrow B^{(j)}C^{(j)}$,$D^{(j)} \leftarrow D^{(j)}C^{(j)}$, $S^{(j)} = \mathrm{diag}(U^{(j)})$
   and $C^{(j)} = I_{\mathrm{NUME}}$. Set $P =$NUME.
5. For $h = 1, \ldots, P_u$ do steps 5.1 to 5.2
   5.1 Find residual Res_{i_h} and compute its norm.
   5.2 If $\|Res_{i_h}\| <$ CRITR, do not include Res_{i_h} in the basis and continue with the

next h. Otherwise, it will be added it the basis. If MBL vectors have been added or LIM has been reached go to step 6.

6. Set P_a to the number of unconverged residuals found in step 5. Apply the diagonal operator on the P_a residuals. Orthonormalize and repeat if necessary. Add the new P_a vectors to B.
7. Augment $D^{(j)}$ by matrix-vector multiplies with the new basis vectors. Augment $S^{(j)}$. by P_a number of columns. Set $P = P + P_a$.

4 The Program

4.1 Subroutines and Libraries

The current version of the Davidson method has been implemented in Fortran 77 on a Sun UNIX operating system. It does not include extensions or operating system calls and therefore should be portable to a wide variety of machines. The code has also been tested on an Intel iPSC/860. Implicit typing to double precision is used for all floating point variables. Therefore, some minimal effort is required to port the code to 64-bit architectures with only single precision.

The program consists of nine subroutines, listed in a calling order: DVDSON, SETUP, ADDABS, DVDRVR, TSTSEL, MULTBC, OVFLOW, NEWVEC, ORTHNRM. Figure 1 shows the calling sequences. It also requires a user specified routine, OP, that performs the matrix block-vector multiplication. The description of these routines follows.

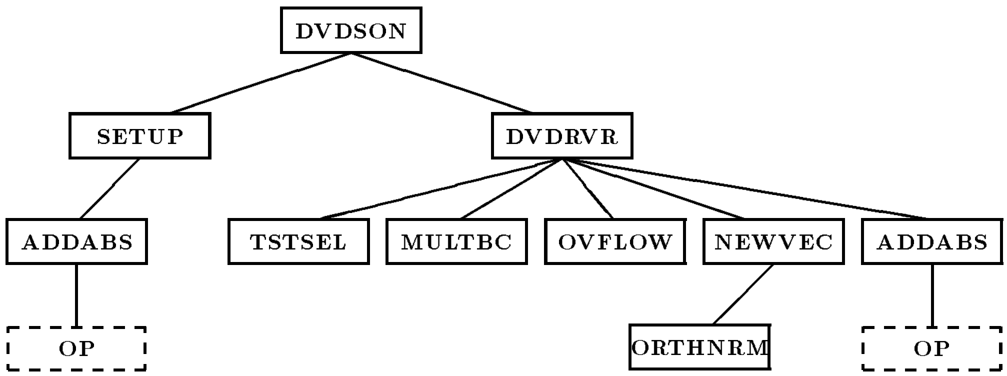

Figure 1: Calling sequence of the DVDSON routines. The order is from higher to lower nodes in the tree, and from left to right. ADDABS is called by both SETUP and DVDRVR routines. The user supplied routine OP is called by ADDABS.

DVDSON: This is the front-end routine called by the user. It performs error checking, initialization of the program parameters, assignment of the internal work arrays, and places the results in the proper place. It calls SETUP and DVDRVR.

SETUP: Performs step 0 of the algorithm. If no initial estimates are specified it sorts the NUME lowest diagonal elements and creates B. It calls ADDABS.

ADDABS: This routine calls OP, to compute $D = AB$ from the new columns of B. It also computes the new columns of S from the new columns of D and B.

DVDRVR: This is the driver routine that implements the principal loop of the algorithm. It calls routines TSTSEL, MULTBC, OVFLOW, NEWVEC, ADDABS. At the beginning of each loop the small eigenproblem is solved (step 1) by a call to DSPEVX, a LAPACK library routine mentioned later in this section.

TSTSEL: Called after the solution of the small problem. Implements steps 2 and 3 of the algorithm.

MULTBC: The routine multiplies the matrix B (or D) with C and stores the result in B (or in D). It is used for contracting the basis B and the auxiliary matrix $D = AB$ in step 4.

OVFLOW: This routine carries out the rest of the step 4 of the algorithm. It computes the restarting S and C.

NEWVEC: Computes the new vectors to be included in the basis (steps 5 and 6). The residuals are found, the convergence is tested and the diagonal operator is applied. Finally, orthonormalization is performed by calling ORTHNRM.

ORTHNRM: Orthogonalizes a sequence of vectors to a basis B and to each other, repeats if necessary, and normalizes them.

In addition to the above the user must provide a routine that performs a matrix block-vector (matrix-matrix) multiplication. This should be specified as external in the calling program and passed as parameter in the DVDSON routine. If OP is the name of this routine, its calling sequence should abide by the following declaration:

SUBROUTINE OP(N,M,B,C)

where B and C are arrays of dimension (N,M). The matrix A is multiplied with B and the result is stored in C. Obviously, the storage format and the elements of A should be passed into OP through COMMON or a file. An OP optimized for the example sparse structure proposed in the following sections is provided along with the program.

The program makes some calls to routines from widely-used libraries, namely BLAS and LAPACK [24, 25, 26]. Several reasons suggest this:

Portability The routines from these libraries are written in standard Fortran 77, and they can run on any machine with Fortran 77 compiler.

Accessibility These libraries are available at no cost from Netlib. Moreover, machine vendors (Cray, Intel, etc.) include most of these library routines (mainly BLAS) in their language libraries, optimized for their architecture.

Efficiency Routines from BLAS and LAPACK use efficient Fortran codes, and the state of the art in numerical algorithms and error analysis.

Vectorizing Supercomputer and Vector Processor vendors have optimized the BLAS routines for vector execution and they provide the libraries along with the machines.

Concise code Using these libraries the Davidson program is shorter, easier to understand, profile, compile and execute.

The program calls the following BLAS routines: DINIT, DDOT, DCOPY, DAXPY, IDAMAX, DSCAL, DGEMV. The driver routine DSPEVX is called from LAPACK library, in order to solve the small symmetric eigenvalue problem. The BLAS and LAPACK routines called from both the program and DSPEVX are shown in Table 1. Routine DINIT, is part of the AUXBLAS library. This routine is provided with the package in case AUXBLAS is not generally available. Finally, it should be mentioned that routine DLAMCH is called by several LAPACK routines, in order to find the double precision parameters of the machine. Once these parameters are known they should be substituted for the routine calls.

BLAS	idamax	lsame	dasum	daxpy	dcopy	ddot	dgemv	dger
	dinit	dmach	dnrm2	dscal	dspmv	dspr2	dswap	xerbla
LAPACK	ilaenv	dlae2	dlaebz	dlaev2	dlagtf	dlagts	dlamch	
	dlansp	dlapy2	dlarf	dlarfg	dlarnv	dlartg	dlaruv	
	dlasr	dlassq	dlazro	dopgtr	dopmtr	dorg2l	dorg2r	
	dspevx	dsptrd	dstebz	dstein	dsterf	dsteqr		

Table 1: Routines required by DVDSON.

4.2 Input-Output

DVDSON is called with the following sequence of arguments:

```
    CALL DVDSON(OP,N,LIM,DIAG,
:             ILOW,IHIGH,ISELEC,NIV,MBLOCK,
:             CRITE,CRITC,CRITR,ORTHO,MAXITER,
:             WORK,IWRSZ,IWORK,IIWSZ,
:             HIEND,NLOOPS,NMV,IERR)
```

The arguments in the list follow the conventional ordering starting with input, continuing with scratch and ending with output. The complete description follows:

OP (Input) The user supplied subroutine for multiplying the matrix with a block of vectors. It should be declared external in the calling routine.

N (Input) The order of matrix A.

LIM (Input) The upper limit on the order of the expanding basis. NUME<LIM≤N must hold, where NUME is defined in the section (3). The case LIM=NUME is allowed only for LIM=NUME=N. It depends on the available worksize. As discussed earlier, it is preferable for LIM to be large but not too large so that the time per iteration is low, and numerical errors are avoided. For atomic structure calculations a number between 20 and 40 larger than NUME has proved efficient.

DIAG (Input) Vector of length N storing the diagonal elements of the matrix A.

ILOW (Input) The index of the lowest eigenpair to be computed from a range of eigenpairs. If ILOW≤ 0 or ILOW>N, the selected eigenpairs to be computed should be contained in array ISELEC. Modified on exit.

IHIGH (Input) The index of the highest eigenpair to be computed from a range of eigenpairs. Considered only when 1≤ILOW≤N. Modified on exit.

ISELEC (Input) Array of size LIM, holding the user specified indices for the eigenpairs to be computed. Considered only when ILOW≤ 0 or ILOW>N. The program looks for the indices starting from the first position of ISELEC until it reaches a non positive integer. The indices can be listed in any order. Modified on exit.

NIV (Input) Number of initial eigenvector estimates provided by the user. If NUME ≤ NIV ≤ LIM, the first NIV columns of size N of WORK are assumed to contain the estimates (see below). In all other cases of NIV the program computes the appropriate number of initial estimates based on DIAG.

MBLOCK (Input) Maximum size of the block of vectors to be targeted in each iteration. $1 \leq MBLOCK \leq K$(the No. of eigenpairs wanted) should hold. A large block-size reduces the number of iterations (matrix accesses) but increases the matrix-vector multiplies.

CRITE (Input) Convergence threshold for eigenvalues. The test is performed on the absolute value of the difference between two consecutive eigenvalue approximations. In the algorithm notation, if $|U^{(j+1)}_{I_h I_h} - U^{(j)}_{I_h I_h}| <$ CRITE for all required eigenvalues $(h = 1, \ldots, K)$ convergence is signaled.

CRITC (Input) Convergence threshold for coefficients. If the coefficient RL$_h$ < CRITC the corresponding eigenvector is considered to have converged. If this occurs for all unconverged eigenvectors, convergence is signaled.

CRITR (Input) Convergence threshold for residual vector norms. If a targeted residual norm is less than CRITR the corresponding eigenvector is considered to have converged. If this occurs for all residuals of unconverged eigenvectors, convergence is signaled.

ORTHO (Input) Threshold above which loss of orthogonality is assumed. The process can be skipped by setting ORTHO to a large number, for example 1.D+3.

MAXITER (Input) Upper bound on the number of iterations of the algorithm. A typical MAXITER can be MAX(200,NUME*40), but it can be increased as needed.

WORK (Input-Output) Double precision array of size IWRSZ. If NUME≤ NIV ≤ LIM, on input, WORK must have the NIV initial estimates. These are vectors of size N, starting from WORK(1) and continuing one after the other. They must form an orthonormal basis.
On exit, WORK stores the eigenvectors, eigenvalues, the eigenvalue differences between the last two steps and the residuals in this order. A description follows:

WORK(1): The first NUME*N locations contain the approximations to the NUME extreme eigenvectors. If the lowest eigenpairs are required, (HIEND=false), eigenvectors appear in ascending order, while for the highest (HIEND=true) they appear in descending order. If only some are requested, the order is the above one for all the NUME extreme eigenvectors, but convergence has been reached only for the selected ones. The rest are the current approximations to the non-selected eigenvectors, in case they are needed for future calculations.

WORK(NUME*N+1): The next NUME locations contain the approximations to the NUME extreme eigenvalues, corresponding to the above NUME eigenvectors. The same ordering and convergence status applies here as well.

WORK(NUME*N+NUME+1): The next NUME locations contain the values of $|U_{ii}^{(j+1)} - U_{ii}^{(j)}|, i = 1, \ldots,$ NUME.

WORK(NUME*N+NUME+NUME+1): The next NUME locations contain the corresponding residual norms to the NUME above eigenvectors.

IWRSZ (Input) The size of the double precision workspace. It must be at least as large as:

(2*N + LIM + NUME+10)*LIM + NUME

IWORK (Input) Integer work array of size IIWSZ. Modified on exit.

IIWSZ (Input) The size of the integer workspace. It must be at least as large as:

6*LIM + NUME

HIEND (Output) Logical variable. If `.true.`, on exit, the highest eigenpairs are arranged in descending order. Otherwise, the lowest eigenpairs are arranged in ascending order.

NLOOPS (Output) The number of iterations it took the algorithm to reach convergence. This is also the number of matrix references.

NMV (Output) The number of Matrix-Vector (M-V) multiplies. Each matrix reference can have up to size(block) M-V multiplies.

IERR (Output) An integer denoting the completion status:
IERR = 0 denotes normal completion.
IERR $=-k$ denotes error in DSPEVX (k eigenpairs not converged)
0<IERR≤2048 denotes some inconsistency as determined by the bits in the binary representation of IERR:
IERR=1 if N < LIM
IERR=IERR+2 if LIM < 1
IERR=IERR+4 if ISELEC(1)<1, and no range specified
IERR=IERR+8 if IHIGH > N (in range or ISELEC)
IERR=IERR+16 if IHIGH < ILOW (Invalid range)
IERR=IERR+32 if $K >$ LIM (Too many wanted)
IERR=IERR+64 if duplication in ISELEC
IERR=IERR+128 if NUME > LIM
IERR=IERR+256 if MBLOCK is out of bounds
IERR=IERR+512 if IWRSZ or IIWSZ is not enough
IERR=IERR+1024 if orthogonalization failed
IERR=IERR+2048 if NLOOPS > MAXITER

5 Memory and Time Requirements

5.1 Memory

After the above description it is possible to express the program's memory requirements as a function of certain input variables. A general formula catering for various problems and architectures is given below:

$$\begin{aligned}
\text{Memorywords} &= \mathit{sizeof}(\text{WORK}) + \mathit{sizeof}(\text{IWORK}) + \\
&\quad \mathit{sizeof}(\text{DIAG}) + \mathit{sizeof}(\text{ISELEC}) + \\
&\quad \mathit{sizeof}(\text{Matrix Commons}) + \\
&\quad \mathit{sizeof}(\text{work vectors required by OP}) + \\
&\quad \mathit{sizeof}(\text{executable code}) + \\
&= N(2\text{LIM} + 1) + \text{LIM}^2 + (\text{NUME} + 17)\text{LIM} + 2\text{NUME} + \qquad (5)\\
&\quad \mathit{sizeof}(\text{OP data}) + \mathit{sizeof}(\text{executable code}) \qquad (6)
\end{aligned}$$

In this equation the size of the matrix as well as any extra arrays that the routine OP uses can be varied according to the user's needs and the memory availability. In one extreme case a full storage is provided for N^2 elements, while on the other extreme no storage is required since the matrix elements are recomputed (or read from disk) each time. Since the *sizeof*(OP data) does not affect the memory requirements of DVDSON it will not be considered hereafter.

The second part of line (6) in the above equation consists of the size of the executable code. The binary for DVDSON compiled in a SUN SPARCstation ELC, is 12380 bytes. When linked with the BLAS and LAPACK libraries compiled for the same machine, the size increases accordingly. The required LAPACK library routines plus the BLAS library compiled on the same machine yield a 229636 bytes binary file.

Part (5) of the above equation shows that the memory space grows linearly with the order of the matrix, N. As mentioned earlier, LIM is usually in the order of a few tens. However, in cases where N is prohibitively large, the program can work with LIM=NUME+1. If NUME=1, only three vectors of size N are needed in the memory at any time. It has been observed that convergence becomes very slow if LIM<NUME+4, therefore it is beneficial to accommodate for two or three more vectors in some applications. Figure 2 graphs the size of the matrix solvable using DVDSON, when the basis is allowed to grow up to LIM. The four different curves correspond to four different memory configurations where all the memory is used for DVDSON excluding matrix representation. Clearly a workstation with 64 Mb RAM can attack problems of size close to 800,000.

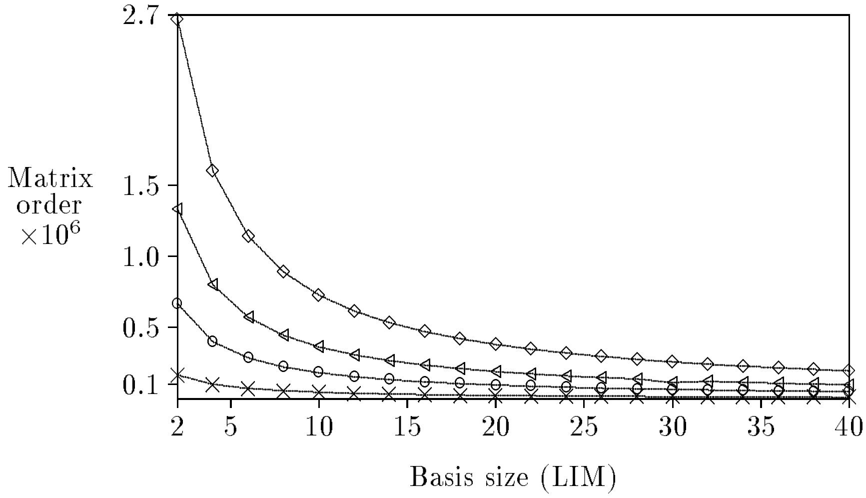

Figure 2: Solvable matrix order as size of basis grows under various memory configurations. ($\times$) 8 Mbytes, ($\circ$) 32 Mbytes, ($\triangleleft$) 64 Mbytes, ($\diamond$) 128 Mbytes.

5.2 Time

Providing meaningful timing measurements for DVDSON is difficult because the execution time depends on the specific machine, the matrix-vector multiplication implementation and the different rates of convergence of different problems. Therefore the number of iterations is considered a more objective "timing". The following results are sampled from a set of matrices, generated by the MCHF and GRASP^2 [5, 7] atomic structure packages. Table 2 gives the basic characteristics of these matrices.

Size	Program	Description	Nonzero	density
410	(MCDF)	Lithiumlike Uranium.	31087	36.9%
748	(MCDF)	All single and double excitations for Oxygenlike Argon.	59659	21.3%
862	(MCHF)	Li, 2S; by n method; n=6.	120735	32.5%
2149	(MCDF)	A complete active space calculation with n up to 4 for Berylliumlike Xenon.	335416	14.5%
8765	(MCHF)	An active space calculation (n=6) for Beryllium. The only matrix elements included in $(A_{ij}), i \geq j$ are $j \leq 25$ if $j \neq i$.	505599	1.3%

Table 2: Description of the test cases.

In Table 3, the number of iterations required for convergence is given. Results from the Netlib Lanczos — **DNLASO**, version: June 1983 — program are recorded as well for the ease of comparisons. Both programs stop with an eigenvalue threshold of 10^{-11}. The lowest eigenpair is required in both methods. Since the matrices are very small, they are kept in memory and no blocking is in effect. Usually ten to thirty iterations are enough for the non-block DVDSON to converge. This has also been noticed by Weber at al. [20]. In the current examples, even though the matrices involved are only moderately sparse, the DVDSON program outperforms the Lanczos one. Block-Davidson requires less iterations to converge, but since the sizes of the current matrices are not large enough to account for disk storage, results are not included here. Davidson [27] gives several examples that demonstrate the usefulness of the block method.

Matrix size	Lanczos		Davidson	
	Iterations	Time (sec)	Iterations	Time (sec)
410	20	2.25	5	1.06
748	42	9.77	9	3.63
862	94	45.53	27	22.20
2149	78	97.32	15	32.95
8765	166	377.66	29	130.94

Table 3: Comparison of the Lanczos and Davidson programs by the number of iterations and time for the matrices of table 2. Eigenvalue threshold: 10^{-11}.

Convergence can be also demonstrated by the decrease of the residual norm in successive iterations. Figure 3(a) plots the residual norm, as it decreases through iterations for each of the five cases. It can be seen that convergence can be extremely rapid, supporting the claim that Davidson provides a second order convergence near the solution. Similarly, Figure 3(b) shows the decrease in the distance between two eigenvalue approximations. Although the pattern is the same as in Figure 3(a), the eigenvalue convergence precedes the residual one by several orders of magnitude. The reason is that the error in the eigenvalue is quadratic in the error in the eigenvector, as in any symmetric variational scheme [28]. Figure 3(c) shows the convergence of the eigenvector components as $\log_{10}\Delta x$, where $\Delta x = \min(\|x^{(j-1)} - x^{(j)}\|_\infty, \|x^{(j-1)} + x^{(j)}\|_\infty)$. The number of iterations is similar to that of the residuals but the components are accurate to machine precision.

The time taken by an iteration for a specific problem size, depends on the current size of the expanding basis, the size of the block entering the basis and the OP routine. The OP routine has a complexity $O(P_a N^2)$ where P_a is the number of vectors added to the basis. In addition, in the notation of the algorithm of section (3) each step involves:

$$O(P^3\mathrm{NUME}^3) + N(4(f+1)\mathrm{LIM} * \mathrm{NUME} + 10P_aP + 7P_a + 3P_a^2) \tag{7}$$

floating point operations, where $O(P^3\mathrm{NUME}^3)$ is the time complexity to diagonalize the small problem, P is the current size of the basis and f is the frequency of restarting.

Finally in Table 3, timings of the method are also given in contrast to timings from the Lanczos one. Both executions were timed on a SPARCstation ELC with 24 Mb memory configuration. The sparsity structure and matrix-vector multiplication presented in the following sections are used in this current program. The method is clearly fast enough to remove the bottleneck from the eigensolution problem to the generation of the matrix [7, 5].

6 A Model for Sparse, Symmetric Matrix Representation

A large number of todays' applications involve matrices of order less than 10^6, and therefore it is most common for the matrix to be stored on disk or in memory for the smaller applications. A sparse data structure is needed for both disk and memory since the matrices are large, symmetric and

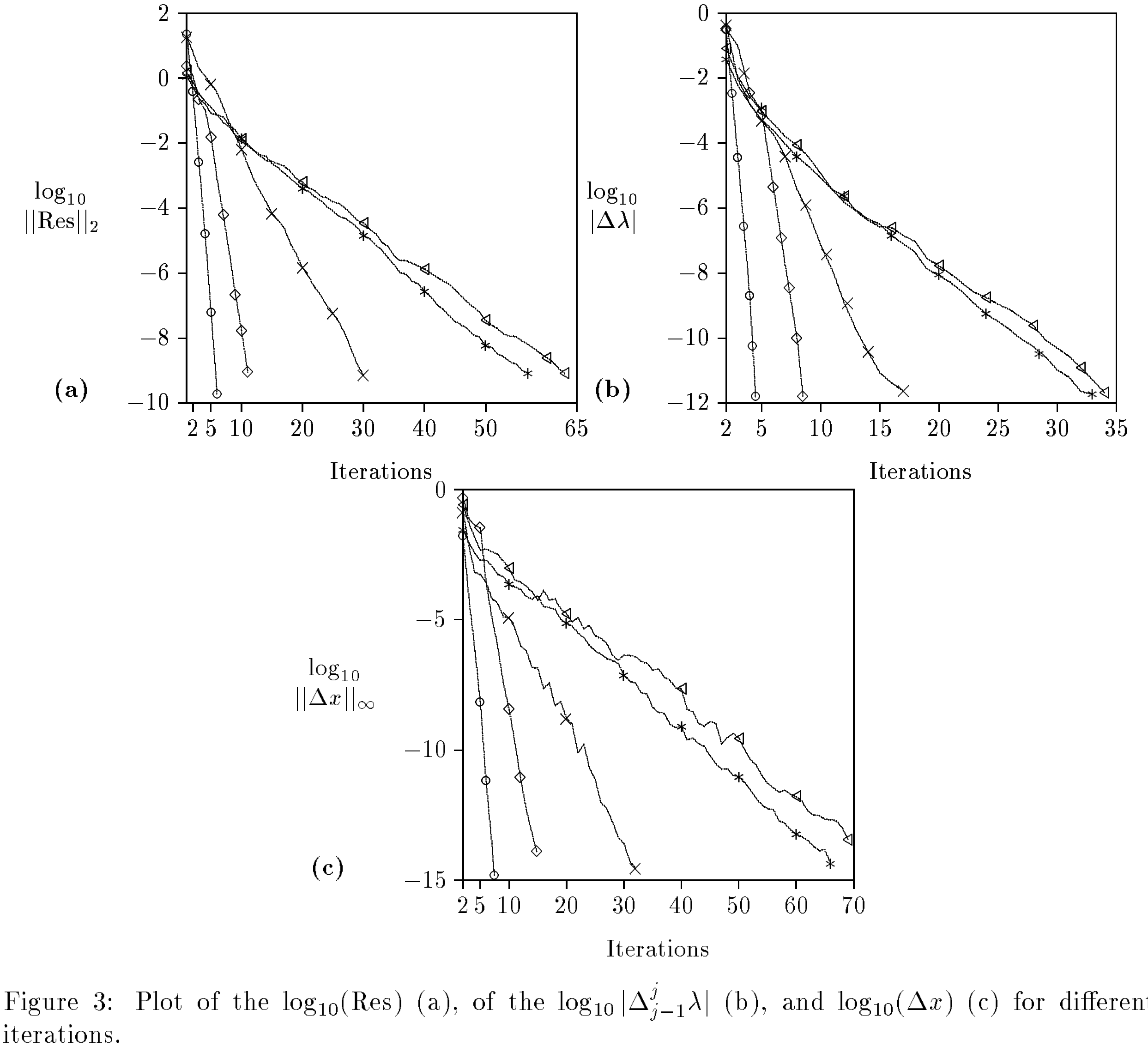

Figure 3: Plot of the $\log_{10}(\mathrm{Res})$ (a), of the $\log_{10}|\Delta_{j-1}^{j}\lambda|$ (b), and $\log_{10}(\Delta x)$ (c) for different iterations.

($\circ$) 410, ($\diamond$) 748, ($\times$) 2149, ($*$) 862, ($\triangleleft$) 8765.

have a substantial fraction of their elements equal to zero. When the sparsity pattern is structured (banded, block, etc.), many storage optimizations are possible because the location of each element in the sparse data structure implies its location in the original matrix. Storage schemes for these cases have been extensively developed and widely used in libraries like BLAS[24, 25].

However, in many scientific calculations, the locations of the non-zero elements are scattered throughout the matrix in an unpredictable way. In these cases a sparse data structure should also keep information of the non-zero element locations. This is usually handled in terms of indices. Various strategies exist for maintaining this data, all aiming at reduced storage and a fast matrix-vector multiplication, non-the-less each serving different needs. Some methods compromise the storage efficiency to get better matrix-vector multiplication performance and vice versa. Sophisticated techniques have been developed to combine the benefits from both goals [29].

A simple, yet very efficient storage scheme, is the compressed sparse row or column format and it is based on the array representation of the adjacency list of a graph [30, 29, 31]. Since the matrix is symmetric only half of the non-zero elements are kept. Keeping the upper part of the matrix by columns is equivalent to keeping the lower part by rows and vice versa. In the current representation the lower part is kept by columns, but it is essentially the same representation as for the upper part.

The adjacency list describing the undirected graph derived from the above matrix can be stored in array format requiring two additional indices. The arrays used for this purpose are described below:

- One dimensional array, ELM, keeping the values of the non-zero elements under the main diagonal for each column. If $i < j$, column i is stored before column j. The size of ELM is limited to the number of non-zero elements of the lower triangular part of the matrix, including the diagonal.

- One dimensional array, IROW, keeping for each of the elements of ELM an index to its row position. Obviously the size of this integer array is the same as the size of ELM.

- One dimensional array, ICOL, keeping an index to where each column ends in the array ELM. The size of this array is N.

The adjacency list data structure saves a considerable amount of storage space compared to many proposed models in the literature [30]. Since for many problems this structure incurs short vectors, it has been reported to result in inefficient matrix-vector multiplication implementations in high performance computers [29, 32]. However, in numerous problems the matrices are dense or large enough to provide long vectors for efficient vectorization or pipelining. Large matrices in atomic structure calculations fall into the latter category, and the above structure has proved efficient for them (see section (5)). An example of such a representation appears in Figure 4.

7 A Matrix-Vector Multiplication Procedure

The drawback for efficiently storing the matrix in a sparse format is the performance degradation in the matrix-vector multiplication. Most sparse schemes suffer from the same problem, because optimized level 2 and 3 BLAS routines cannot be used since the matrix is not fully stored in an array. Moreover, since adjacently stored elements are not necessarily adjacent in the matrix, level 1 BLAS routines cannot be used either.

Based on the above sparsity structure, a routine for performing multiplication of the matrix with a block of vectors is presented. This routine uses "gather-scatter" routines to place in adjacent

$$A = \begin{pmatrix} 5. & & & & & \\ 6. & 7. & & & & \\ 3. & 0. & 1. & & & \\ 0. & 0. & 8. & 10. & & \\ 11. & 12. & 0. & 0. & 13. & \\ 20. & 21. & 0. & 17. & 16. & 22. \end{pmatrix}$$

ELM	5.	6.	3.	11.	20.	7.	12.	21.	1.	8.	10.	17.	13.	16.	22.
IROW	1	2	3	5	6	2	5	6	3	4	4	6	5	6	6
ICOL	5	8	10	12	14	15									

Figure 4: The arrays ELM, IROW and ICOL consist the sparse representation of the symmetric matrix A.

locations those elements of the block of vectors that are involved in a calculation with the current column. This enables the use of level 1 BLAS routines, and thus offers the possibility of optimized and vectorizable codes as it has been mentioned in the literature [10, 30, 33]. An approach where small subblocks are built for use of level 2 BLAS, is impeded by the storage only of the lower triangular part and further by significant additional memory requirements.

Apart from setting the basic calculation block of the matrix to be the column, the current implementation is based on the following:

- In a matrix-vector multiplication each of the matrix columns contributes some part to the final result. When the matrix is multiplied with a block of vectors, each matrix column records its contributions to all the vectors, before the next column is examined. In this way only one pass through the matrix is necessary which is vital when it is stored on disk. It also allows for locality benefits, if the column can fit in cache, and further if a parallel computer is used, more work can be assigned to each processor in parallel (e.g., Cray).

- The lower triangular form of the matrix suggests that each stored column i of the matrix A contributes to the result in two ways: as the column of the lower triangular part of the matrix (**DAXPY**), and as the row of the upper triangular one (**DDOT**). If d is the result vector, the contributions are:

$$d_j = d_j + A_{j,i} b_i, \quad \forall j = i+1, \ldots, N \quad (DAXPY), \tag{8}$$

and

$$d_i = d_i + \sum_{j=i}^{N} A^T_{i,j} b_j. \quad (DDOT) \tag{9}$$

- In equations (8) and (9) the elements of vectors b and d involved in the calculation are those that correspond to the non-zero elements of column i. To use **DDOT** and **DAXPY**, these elements of b and d must be in adjacent memory locations. The gather routine is used for this purpose. Clearly, after the end of the calculation the results should be scattered back to their original positions.

- The routine has been designed to cater to either upper or lower storage format depending on the user specified logical variable LUPPER; if `.true.` the matrix is kept as columns of the upper triangular part otherwise as columns of the lower triangular part.

- The argument list is the same with the routine OP required by DVDSON. The matrix, indices, the logical variable LUPPER, and temporary arrays are passed through the following commons:

```
COMMON /MATRIX/  ELM(NONZER),ICOL(N),IROW(NONZER),LUPPER
COMMON /TEMP/ TEMPB(N),TEMPC(N)
```

The price paid for the use of **DDOT** and **DAXPY** through gathering and scattering, is the additional storage requirement of two arrays of length equal to the maximum number of nonzero elements in a column. These keep the temporary gathered values of b and d. The same arrays can be used for different vectors in the case of block of vectors.

Along with a memory version of this routine, called **DSSBMV** (double precision sparse symmetric block matrix-vector), the two routines **GATHER**, **SCATTER** are provided with the DVDSON program. **DSSBMV** has been tested with all the cases of section (5.2). The timings in Table 3 reflect its effectiveness. A disk version is easily obtained by a customization to the user's disk system and disk storage.

8 Using the code

A sample test

A simple case is given for installation purposes in the sample driver program provided with the DVDSON package. The matrix is a banded symmetric one, of order $N = 100$ and band size under the main diagonal, iband=10. All the off diagonal elements are set to 0.001, and the diagonals to their column index. The eigenvalues of the matrix are the slightly perturbed diagonal values. Four test cases are tried on one run:

1. In the first case the lowest NUME=10 eigenpairs are found. No initial estimates are available and the residual convergence threshold is large so that rough approximate estimates can be found in a few steps.

2. In the second case, the same range of eigenpairs is required, but the thresholds now are tight, and the approximations from the previous step are used as initial estimates.

3. In the third case only selected eigenpairs from the highest end of the spectrum are required, specifically eigenpairs with indices $N, N-5, N-9$. No initial estimates are specified. Moreover, block and reorthogonalization is switched on by setting MBLOCK=3 and ORTHO to a small value. Since NUME=10, 10 eigenpairs are listed but convergence has been reached only for the required ones.

4. Finally, in the fourth run the above initial estimates are used to find the 10 highest eigenpairs. Now the block size is set to MBLOCK=NUME=10. This case offers an example of the necessity of reorthogonalization. A large ORTHO inhibits residual convergence below 10^{-6}. A value of ORTHO= 10^{-9} was necessary for the method to converge for CRITR= 10^{-10}.

The results appear at the end of the paper.

Method limitations

The DVDSON program will usually converge whenever the Lanczos converges. However, in cases where the matrix is permutationally equivalent to a block diagonal one, DVDSON may converge to the wrong eigenpair. Each diagonal block defines an invariant subspace of the space defined by A. DVDSON selects the initial vectors by attempting to guess the dominant component of the eigenvector. If the resulting unit vector happens to be in the wrong invariant subspace, convergence will be satisfied for the lowest eigenpair of that subspace. To solve the problem, the independent blocks of the matrix have to be identified, and DVDSON applied on each of them. This is necessary because for an initial vector with overlap to all invariant subspaces, it is possible that a higher eigenpair converges faster than the extreme ones. A solution to this problem is developed in [34].

A similar problem could occur when the initial estimate is orthogonal to the eigenvector required or reflects the structure of some other eigenvector. This is particularly the case when the diagonal elements of the matrix are equal and thus the method chooses arbitrary unit vectors as initial estimates. Convergence in this situation can be very slow [35]. To partially remedy this problem more eigenpairs than needed should be asked for and a large block size should be used.

9 Conclusions

The DVDSON program that finds a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix has been described. The program is based on a version of the Davidson algorithm which offers convergence improvement over the Lanczos algorithm. The extensions followed in the current program are described and the improvements on the algorithm's flexibility and efficiency justified. A brief users' manual is given with a description of the routines and libraries of the program, as well as a detailed presentation of the input and output. The code depends neither on some specific storage of the matrix nor on a specific matrix-vector multiplication. For completeness, an example sparse data structure is proposed and an efficient, vectorizable matrix-vector multiplication routine is developed based on this data structure.

In 1990, the record of largest matrix size solved was in $O(10^9)$ in quantum chemistry, by using a modified version of the Davidson algorithm [36]. The choice of this algorithm was based both on its memory and its convergence characteristics. The purpose of the current program is to provide a general purpose Davidson program that has the same power characteristics for smaller size of problems (up to 10^6), appearing in many fields like atomic structure calculations.

Acknowledgements

This work has been supported by a National Science Foundation grant No. ASC-9005687. One of the authors (A. Stathopoulos) would like to thank Farid A. Parpia for sharing his experience and for his support.

References

[1] B.N. Parlett, SIAM J. Sci. Stat. Comput. 5 (1984) 590.

[2] A. Jennings, in: Sparse Matrices and Their Uses, ed. I. Duff (Academic Press, New York, 1981) p. 109.

[3] E.R. Davidson, Comput. Phys. Commun. 53 (1989) 49.

[4] C.W. Bauschlicher and P.R. Taylor, J. Chem. Phys. 85 (1986) 2779.

[5] C.F. Fischer, Comput. Phys. Commun. 64 (1991) 369.

[6] C.F. Fischer, Comput. Phys. Commun. 64 (1991) 473.

[7] F.A. Parpia, I.P. Grant and C.F. Fischer, 1990.

[8] B.S. Garbow et al., Matrix eigensystem routines : EISPACK guide extension, (Berlin ; New York : Springer-Verlag, 1977).

[9] HARWELL Subroutine Library (Release 10), (Theoretical Studies Department, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, England, 1992).

[10] V.M. Umar and C.F. Fischer, The International Journal of Supercomputing Applications, 3 (1989) 28.

[11] E.R. Davidson, J. Comput. Phys. 17 (1975) 87.

[12] C. Lanczos, J. res. Nat. Bur. Stand. 45 (1950) 255.

[13] C.C. Paige, J. Inst. Math. Applic. 10 (1972) 373.

[14] R.B. Morgan and D.S. Scott, SIAM J. Sci. Stat. Comput. 7 (1986) 817.

[15] M. Crouzeix, B. Philippe and M. Sadkane, Tech. Rep., Report TR/PA/90/45, CERFACS, Toulouse, 1990.

[16] E.R. Davidson, in: Methods in Computational Molecular Physics, eds. G.H.F. Diercksen and S. Wilson (Reidel, Dordrecht, 1983) p. 95.

[17] E.R. Davidson, J. Phys. A 13 (1980) 179.

[18] W. Butcher and W.JE. Kammer, J. Comput. Phys. 20 (1976) 313.

[19] B. Liu, in: Numerical Algorithms in Chemistry: Algebraic Methods, eds. C. Moler and I. Shavitt, LBL-8158 Lawrence Berkeley Laboratory (1978).

[20] J. Weber, R. Lacroix and G. Wanner, Comput. Chem. 4 (1980) 55.

[21] Grand Challenges: High Performance Computing and Communications. Office of Science and Technology Policy, 1991. A report by the committee on Physical, Mathematical and Engineering Sciences.

[22] G. Cisneros and C.F. Bunge, Compu. Chem. 8 (1984) 157.

[23] G. Cisneros, M. Berrondo and C.F. Bunge, Compu. Chem. 10 (1986) 281.

[24] C. L. Lawson, R. J. Hanson, D. Kincaid and F. Krogh, ACM Trans. Math. Soft. 5 (1979) 308.

[25] J. J. Dongarra, ACM Trans. Math. Soft. 14 (1988) 1.

[26] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, LAPACK User's Guide, (SIAM, Philadelphia, 1992).

[27] C.W. Murray, S.C. Racine and E.R. Davidson, Tech. Rep. (1991).

[28] B.N. Parlett, The Symmetric Eigenvalue Problem (Prentice-Hall, New Jersey, 1980).

[29] R.C. Agarwal, F.G. Gustavson and M. Zubair, Supercomputing '92 proceedings (IEEE Computer Society Press 1992) p. 32.

[30] B. Philippe and Y. Saad, in: Proceedings of International Workshop on Parallel Algorithms and Architectures, eds. M. Cosnard at al. (North-Holland, Amsterdam, 1989) p. 33.

[31] E. Horowitz and S. Sahni, Fundamentals of Data Structures, (Computer Science Press, Rockville, MD, 1983).

[32] J. Erhel, International Journal of High Speed Computing 2 2 (1990) 101.

[33] J.G. Lewis and H.D. Simon, SIAM J. Sci. Stat. Comput. 9 2 (1985) 304.

[34] F.A. Parpia, A. Stathopoulos and C. F. Fischer, in preparation.

[35] T.Z. Kalamboukis, J. Phys. A 13 (1980) 57.

[36] J. Olsen, P. Jørgensen and J. Simons, Chem. Phys. Lett. 169 (1990) 463.

[37] A. Stathopoulos and C. F. Fischer, Intel Supercomputer Users' Group, 1991 Annual Users' Conference, (1991) 343.

[38] J.H. Wilkinson, The Algebraic Eigenvalue Problem, (Oxford University Press, New York, 1965).