

A New Eigenvalue Solver For Atomic Structure Calculations

Mikhail SAPAROV and Charlotte FROESE FISCHER
Department of Computer Science
Box 1679B, Vanderbilt University
Nashville, Tennessee 37235 USA

email: `misha@vuse.vanderbilt.edu` and `cff@vuse.vanderbilt.edu`

November , 1996

Prepared for the U.S. Department of Energy
under grant number DE-FG02-97ER14761

Abstract

This paper proposes a new method for finding a few extreme eigen pairs of large sparse matrices. The sequential implementation of the algorithm is described. The goal is to test the algorithm on matrices arising in real applications and compare the performance with currently used techniques such as the Davidson method.

1 Introduction

In atomic structure calculations (ATSC) often a few extreme eigenvalues and corresponding eigenvectors of large sparse matrices are desired. Because of the large size of matrices, dense matrix eigenvalue solvers are inappropriate for that task. Since only the extreme eigenpairs are required, it is more effective to use iterative methods. A variety of such methods already exist namely the Power method, Lanczos type methods, and Davidson method which is currently being used in ATSC . The reason for using the Davidson method is its faster convergence rate [1, 2] for diagonally dominant matrices which frequently are the case in ATSC. However, the latest calculations done in that field showed that the convergence of Davidson method, though faster than any of the other existed methods, is unaffordably slow, and thus requires a new approach to the problem. Such an approach, suggested by Fischer [3], is analyzed in this paper.

1.1 Davidson method

Let's first take a look at the Davidson method (DM) to understand how the new method differs. Davidson was specifically designed for use on large symmetric matrices with strong diagonal dominance. It belongs to the family of the subspace projection methods, which are mathematically equivalent to the Lanczos method [4, 5] but it differs in the way it expands the subspace from the Lanczos. Davidson's idea [1] was that the optimal correction of the eigenvector may be obtained by finding z for which [1]

$$(A - \lambda I)z = -r^{(k)}, \quad (1)$$

where $r^{(k)}$ is the residual. Since we don't know λ and solving the system is expensive, the Davidson method takes the available approximation of λ , λ_k , and exploits the fact that A is diagonally dominant to compute z using the diagonal of A in one step of a Jacobi iteration

$$z = -(diag(A) - \lambda_k I)^{-1} r^{(k)}. \quad (2)$$

Then z is made orthogonal to the basis computed on the previous iterations and the process is continued until convergence is reached. Eq. 1 can be viewed as a preconditioner to the method, and by solving it in different ways a number of modifications of DM have been created [6]. The performance analysis of these modifications in the context of ATSC has been done recently [7] and showed that the best results are given by the Davidson-Olsen method [6] (DO) which will be used for comparison with the new method analyzed in this paper.

1.2 A New Solver

The proposed method utilizes a partitioning scheme rather than Lanczos or Krylov subspace approach. It also relies on a feature of an eigenvalue problem arising in ATSC that location of the large components in the matrix is known and a small matrix can be defined whose eigenvalues are good initial estimates for the corresponding eigenvalues of the full matrix[3]. Then the following partitioning scheme is proposed.

Let A be a large sparse symmetric diagonally dominant matrix of size N . Let A^{00} be a small matrix of size n , whose extreme eigenvalues are good initial estimate of the full matrix. Suppose that A is partitioned so that

$$A = \begin{bmatrix} A^{00} & A^{10^T} \\ A^{10} & A^{11} \end{bmatrix}. \quad (3)$$

Let v be an eigenvector of A , partitioned as follows

$$v = \begin{bmatrix} x \\ y \end{bmatrix}, x = v[1..n] \quad (4)$$

and λ a corresponding eigenvalue. Then we can write an equation

$$A^{10}x + (A^{11} - \lambda I)y = 0. \quad (5)$$

Notice that the eigenvectors $u^{(i)}$ of A^{00} form a basis for x so that $x = \sum_{i=1}^n c_i u^{(i)}$. Also, the matrix $(A^{11} - \lambda I)$ is symmetric and will be positive definite if λ is below all eigenvalues of A^{11} . Therefore, if good estimates of λ and x are available, y can be found by solving Eq. 5.

1.3 Algorithm for finding the k 'th eigen pair

1. Form the small matrix A^{00} and find all its eigenvalues θ_j and eigenvectors u^j .
2. Let $x^0 = u^k$, $\lambda_0 = \theta_k$.
3. Form the basis

$$B = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad (6)$$

4. Until converged

- (a) Solve

$$(A^{11} - \lambda_i I)y^{(i)} = -A^{10}x^{(i)} \quad (7)$$

for $y^{(i)}$.

- (b) Augment the basis B with vector $b^{n+1} = \begin{bmatrix} 0 \\ y^i \end{bmatrix}$, normalized

- (c) Form the projection $S = B^T A B$

- (d) Solve the $n+1$ eigenvalue problem $S c = c \lambda_{i+1}$

- (e) Let $x^{(i+1)} = \sum_{j=1}^n c_j u^{(j)}$

2 Implemenation Specifics

The algorithm above does not specify how exactly we should solve Eq. 7. Actually the main part of the computation in the method is done in that step. Thus, the choice of a linear solver is crucial to the algorithm's performance. We have a restriction posed by the large matrix size that is, it is inefficient and usually even impossible to use direct methods because of fill-in problems arising during decomposition, so the choice of solvers is limited to iterative. In general, solving the linear system iteratively may be a very time consuming operation. However, in the case of ATSC this task can be handled very efficiently by carefully analyzing the problem. As it was mentioned before, the matrix $(A^{11} - \lambda I)$ is symmetric and, assuming the k 'th eigenpair is targeted ($k < n$), positive definite. This allows us to use the preconditioned conjugate gradient (PCG) method which is known for its fast convergence properties [8].

2.1 The choice of the preconditioner

Choosing the preconditioner to the PCG is not an easy task since the speed of convergence depends heavily on that choice and so does computational intensity. Traditionally four preconditioners are considered in combination with PCG [9] sorted in order of the growing complexity costs: Jacobi, Gauss-Seidel Iterations(GS), Successive overrelaxation(SOR), and Incomplete Cholesky (IC). GS and SOR are more suitable for block-diagonal and banded matrices A arising in solutions for PDE [8]. This is not the case in ATSC, where A is randomly sparse. Jacobi preconditioning is cheap but the least effective, so IC is the only choice for reaching rapid convergence.

However, IC decomposition is too costly to be done on each CG iteration. By observing the fact that $|\lambda_{i+1} - \lambda_i|$ is usually much less than $|A_{11}^{11}|$ one can do decompositions only when necessary, i.e. when the change in eigenvalue approximation is sufficiently large relative to the first diagonal element of A^{11} . Moreover, if one is looking for a few eigenpairs, say the k 'th and the m 'th it is known from the initial approximation, θ_k and θ_m respectively, that

$$|\theta_k - \theta_m| \ll |A_{11}^{11}|,$$

and there already exists a Cholesky factorization for the matrix

$$A^{11} - \theta_k I,$$

then IC decomposition can be skipped for matrix

$$A^{11} - \theta_m I,$$

hoping that the Cholesky factorization is close enough for these matrices. The experiments discussed later in this paper showed that these assumptions were fair for practical purposes.

2.2 Other computational cuts

Considering the structure of the matrix B

$$B = \begin{bmatrix} u_1^1 & \dots & u_1^n & 0 \\ u_2^1 & \dots & u_2^n & 0 \\ \vdots & \dots & \vdots & \vdots \\ u_n^1 & \dots & u_n^n & 0 \\ 0 & \dots & 0 & y_1^i \\ \vdots & \dots & \vdots & \vdots \\ 0 & \dots & 0 & y_{N-n}^i \end{bmatrix},$$

one can observe that

$$B^T A B = \begin{bmatrix} \theta_1 & & & u^{(1)T} A^{10T} y^i \\ & \ddots & & \vdots \\ & & 0 & \\ 0 & & \theta_n & u^{(n)T} A^{10T} y^i \\ u^{(1)T} A^{10T} y^i & \dots & u^{(n)T} A^{10T} y^i & y^{(i)T} A^{11} y^i \end{bmatrix} \quad (8)$$

is an "almost" diagonal symmetric matrix with a constant $n \times n$ part. It is obvious that we do not have to do an expensive $B^T A B$ multiplication on each step. Also notice that all elements in last row and correspondingly in the last column, except the diagonal element, contains a constant multiple $u^{(i)T} A^{10T}$, and we can compute these vectors just once. Then we convert the operation which was initially of order $O(nN^2)$ complexity to $O(nN)$.

As was mentioned earlier, algorithm can be adapted easily to compute several eigenvalues in one call and it is very beneficial if eigenvalues are close since we can save a lot of computation by skipping IC decompositions.

2.3 Complexity analysis

Let's compare the computational complexity of the algorithm in general with the DO method assuming large sparse matrices. DO's most expensive operation is a matrix-vector multiplication (matvec) which generally should be performed once for each targeted vector on each iteration. Also all Davidson methods require a rather expensive orthogonalization process on each iteration during the expansion of the orthonormal basis. However, performance of the DO method is usually measured in matvecs as an operation of the maximum complexity of $O(N^2)$, where N is the size of the whole matrix.

The proposed algorithm also includes a few matrix-vector multiplication of the same complexity on each iteration required to solve the linear system. But the method also does the Cholesky decomposition where complexity is $O(NZE^{\frac{3}{2}})$, where NZE is number of nonzero matrix elements. Thus, the complexity of the decomposition is related to the degree of sparsity. Also, the number of required Cholesky decompositions depends on the accuracy of the initial estimate and proximity of the targeted eigenvalues. Therefore, in the worst case, the algorithm will perform slower than Davidson; but practically, the matrices

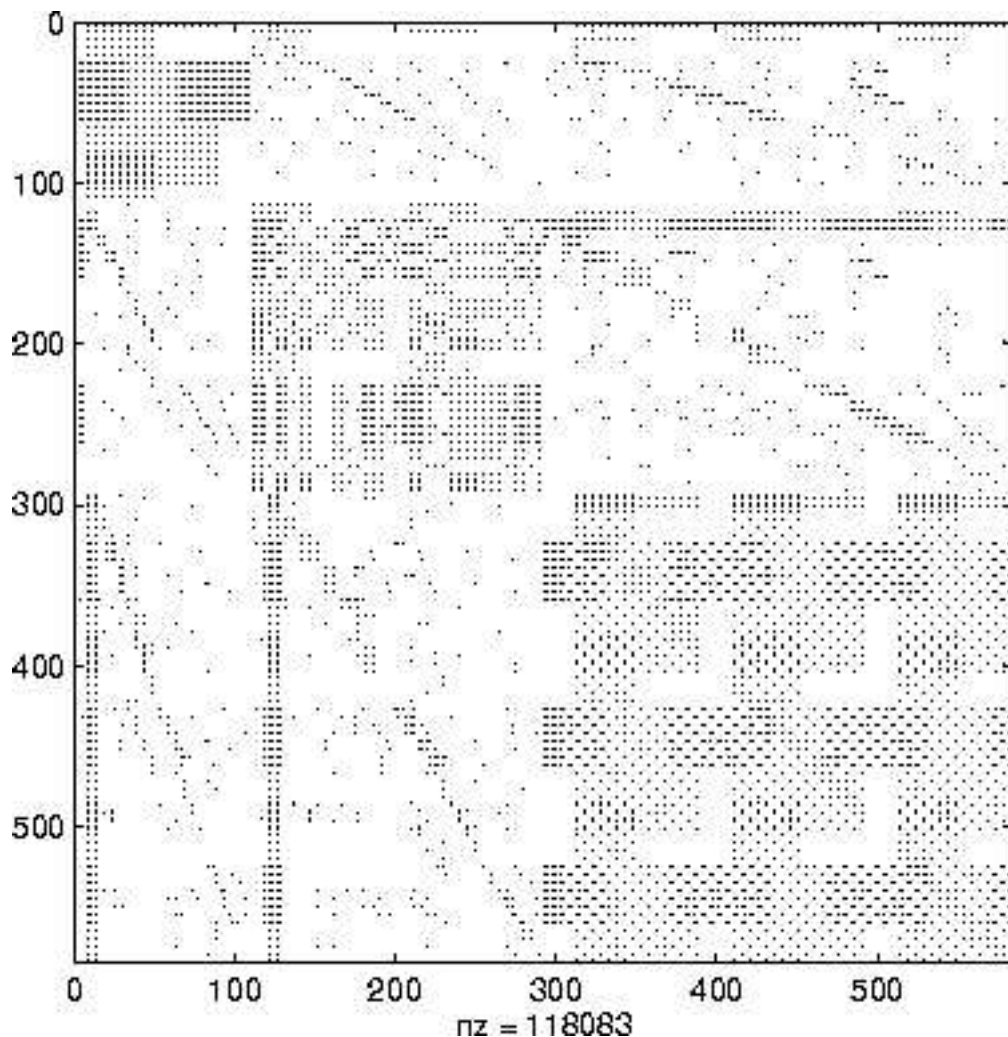


Figure 1: The typical structure of matrix in atomic structure calculations

for ATSC are very sparse, and targeted eigenvalues are relatively close compared with the value of the first diagonal element of matrix A^{11} ¹.

2.4 Choice of the initial estimates

In general, choosing the initial estimate for the eigenvalue problem is not obvious, and most methods use corresponding unit vectors for that purposes. However, the version of DO method used in comparison tests, was tuned up to take advantage of the fact that in ATSC good initial estimates are usually available. The typical element distribution for ATSC is shown in figure 1. As we can see, matrix contains clearly definable blocks. The nice property of ATSC is that all "important" elements are contained in the upper diagonal block. Thus, this block is a natural candidate for the matrix A^{00} in Eq. 3. The same block is used in DO to obtain initial estimates for the targeted vectors.

¹Note that it's not related to degenerated eigenvalues anyhow, indeed all eigenvalues are well separated.

2.5 Linear solver considerations

No matter which iterative linear solver is chosen for the step 4a, there are two parameters controlling convergence of the iteration process: the number of maximum allowed iterations and desirable value of the convergence criterion

$$\delta = \|z^i - z^{i-1}\|,$$

where z^{i-1} , z^i are two consecutive solution approximations. Both parameters effectively influence the number of total matvecs. The accuracy of the linear system solution should be higher than the desired accuracy of the eigenvector. However, in order to save on the number of the matvecs, it is possible to adjust the convergence criterion for the linear system gradually using the following scheme

1. calculate the correction vector y from Eq. 7 with low iterative solver accuracy parameter
2. compute the approximation of the eigenvector, and the absolute difference between current and previous eigenvector values.
3. if relative precision of the current eigenvector is equal to the linear solver convergence parameter δ , let

$$\delta = \delta \times 10^{-2}$$

4. go to 1

All iterative linear solvers also require an initial estimate to start the iteration process. Usually this estimate is set to be the zero vector. It is not the case in this algorithm since the current correction vector y^i is a good initial approximation for y^{i+1} . This observation saves a lot of time because the closer we are approaching to the eigenvector convergence the less correction y^i requires and correspondingly fewer iterations are required for the linear equation solver to converge.

3 Implementation and benchmarks

The actual program was coded as a sequential FORTRAN-77 subroutine. It should be mentioned that the algorithm possesses good parallelization characteristics. Moreover, it's major component, PCG with IC decomposition has been implemented in a number of scientific parallel libraries [10]. However, in this paper only the sequential implementation and benchmarks will be discussed.

The program uses BLAS, LAPACK and Sparse Linear Algebra Package Routines (SLAP) to implement common operations. LAPACK subroutine DSYEV is used on steps 1 and 4d of the algorithm to obtain a solution for the small eigenvalue problem. The SLAP implementation of the PCG method is used on the step 4a to solve the system of the equations. The SLAP matrix format [11] was used to store the sparse matrix in the memory.

Table 1: Recomputing Cholesky factor every iteration vs. computing it just once on the initial iteration.

Case	Matrix Size	No. of Iterations		No. of Matvecs		Time (sec)	
		Once	Many	Once	Many	Once	Many
B1S8	584	5	5	12	11	3.21	5.83
O4P2	2023	8	8	29	26	15.27	78.45
B1Sb	10809	10	10	32	30	353.69	7913.14

The code checks the necessity of rebuilding the Cholesky decomposition on each step with the default threshold

$$\varepsilon < 0.2, \text{ where } \varepsilon = \left| \frac{\lambda_0 - \lambda_i}{A_{11}^{11}} \right|$$

Depending on the estimated complexity of the Cholesky decomposition, the threshold can be adjusted to gain maximum run time efficiency. Assigning zero to threshold, for instance, will give the lowest number of matvecs, but the decomposition will be done on every iteration, while making it large will ensure that the decomposition will be done just once although the number of iterations required to solve the linear system might significantly increase causing more matvecs.

Also, as it was mentioned earlier, it is preferable to target all desired eigen pairs at one call due to the possible significant calculation savings on Cholesky decomposition. However, the number of the targeted eigen pairs is limited by memory constraints. The code requires N DOUBLE PRECISION elements to be allocated for each vector being sought which on most workstations keeps an effective limit of the vectors under a hundred. Another restriction on the number of targeted vectors is that it should be less than the size n of A^{00} because of the algorithm's design.

3.1 Practical Experiments

The tests discussed in this section were done in order to check some assumptions made earlier about the algorithm design, performance, and accuracy. The first test was designed to check the assumption that the eigenvalues are relatively close compared with A_{11}^{11} and Cholesky factor should be recomputed relatively seldom if never.

The second test is run on a matrix of size feasible for solution using LAPACK routine DSPEVX. The results of the the DSPEVX were refined to get the machine precision by using one step of Rayleigh Quotient Iteration method. The absolute error of the eigenvalues and eigenvectors was then calculated. The absolute values of the targeted eigenvalues were around one, and the eigenvectors were normalized. The results in table 2 show the corresponding data.

The third test was done to demonstrate algorithm's stability. The matrix A^{00} was varied in size from optimal in order to observe the algorithm's behavior. It was expected that the algorithm will converge but rather slowly because of the worse initial estimates and non-optimal basis. The results of the experiment proved those expectations.

Table 2: The absolute eigenvector and eigenvalue errors of the algorithm for case B1S8 ($N = 584$) with eigenvector threshold 10^{-8} . The exact solution was obtained by using LAPACK routine DSPEVX and refining it with a one step Rayleigh Quotient Iteration method.

Eigen pair No.	Eigenvalue Error	Eigenvector Error
1	.1107585E-14	.5522570E-10
2	.1258992E-14	.1144714E-09
3	.9381482E-15	.3373893E-09
4	.2291521E-14	.2576852E-10
5	.1070319E-14	.1195177E-09

Table 3: The speed of convergence depending on the size n of matrix A^{00} for case B1S8 ($N = 584$) with eigenvector threshold 10^{-8} . Looking for the lowest eigen pair.

n	matvecs	iter	Eigenvector error
50	51	21	.1198213E-07
60	51	24	.1025090E-07
70	51	24	.1012870E-07
80	33	16	.6403240E-08
90	33	16	.6482586E-08
100	40	20	.8583047E-08
110	11	5	.6654907E-10
111	12	5	.5522570E-10

Table 4: A comparison of the execution time, MATVECS, and number of iterations of DO and proposed algorithm with eigenvector threshold 10^{-8} . Looking for the lowest eigenvalue.

CASE	N	NZE	n	IC time (sec)	time (sec)		matvecs		iter	
					DO	Alg	DO	Alg	DO	Alg
B1S8	584	59334	111	0.78	1.79	3.21	38	12	37	5
O4P2	2023	362511	30	9.20	1.30	15.27	18	29	17	8
B1Sb	10809	5610908	28	254.12	164.49	353.69	135	32	134	10
N1D	18566	11474386	400	18859	14682	31323	38	10	37	4

Table 5: A comparison of the execution time, MATVECS, and number of iterations of DO and proposed algorithm with eigenvector threshold 10^{-8} . Looking for multiple lowest eigenvalues.

CASE	N	NZE	n	IC time (sec)	time (sec)		matvecs		iter	
					DO	Alg	DO	Alg	DO	Alg
B1S8	584	59334	111	0.78	14.83	15.19	180	104	40	28
O4P2	2023	362511	30	9.20	4.08	35.23	41	139	23	27
B1Sb	10809	5610908	28	508.23	591.35	1069.00	411	185	159	57
N1D	18566	11474386	400	18859	91023	81332	188	98	42	27

3.2 Comparison Benchmarks

All benchmarks in this section were done using the ATSP CI code [12] and compared with the DO algorithm with initial estimates provided as described earlier with the new algorithm. Also, when matrix size permits, the solution has been obtained using LAPACK subroutine DSPEVX, and used as an exact solution to estimate the approximation error. The benchmarks were done on the IBM RS6000 /375 with 128MB RAM. Both versions of the program CI were compiled using IBM xlf compiler with optimization option -O2. The timing information reflects the time elapsed from the call to the corresponding subroutine to the moment the subroutine returned control to the calling program. The total time of the CI execution is not considered in the benchmarks. The number of matvecs gives the number of matrix vector multiplications in each case, while number of iterations means the total number of iterations required for the corresponding eigen solver to converge to the desired accuracy.

In table 5, the number of eigenvalues sought was five for B1S8 case, two for O4P2 case, three for B1Sb case, and five for N1D case. The targeting of more than these required the increase of n in cases O4P2 and B1Sb since the initial small matrices had not provided good enough initial estimate.

The proposed algorithm almost always performed worse than DO in sense of physical time. However, it almost always showed better performance in sense of number of iterations and what is more important matvecs, which is usually a good comparison characteristic for eigenvalue problem methods. There were a few reasons for algorithm to perform poorer than DO. First, the only available stable implementation of the conjugate gradient method

Table 6: A comparison of the execution time, MATVECS, and number of iterations of DO and proposed algorithm with eigenvector threshold 10^{-8} for the case B1S8. Looking for the k -th lowest eigenpair.

k	time (sec)		matvecs		iter	
	DO	Alg	DO	Alg	DO	Alg
1	1.79	3.21	38	12	37	5
2	3.86	4.55	84	10	48	5
3	6.15	5.17	111	13	41	6
4	8.31	4.49	159	11	47	5
5	14.83	5.52	180	17	40	7

from the SLAP package was heavily tuned up in favor of vector machines like CRAY X-MP, while the tests were performed on the regular uniprocessor architecture. The alternative routine from the appendix to book [13] showed much better performance but was unstable and could not be used for tests. Also each algorithm's iteration performs the solution of the projected matrix on step 4d. This was done using the standard LAPACK routine DSPEVX which solves the system directly [14]. The execution time for this step can be reduced significantly by considering the special structure of the projected matrix in eq. 8 which is "almost" diagonal by using either specially designed direct method [15] or iteratively. As we can see from the results for N1D calculations in table 5, in case when matvecs take significantly more time than solving the small eigenvalue problem, the proposed method actually does perform faster than DO.

Another point which should be mentioned is that though all test cases were done with the matrix in memory, many large calculations require storage of the matrix on disk. In such case, number of matvecs becomes critical since the complexity of matvecs adds up with the disk I/O time necessary to load matrix into RAM. Thus, it is very advantageous to use the proposed method with "negative" fill IC, that is, if the current calculated element of the Cholesky factorization is less than some threshold value, it is skipped. That way, we can get the size of the IC factorization matrix much less than the size of original matrix, and keep the IC factor in memory, while the original matrix stays on disk.

However, one set of tests should be mentioned separately. The severe disadvantage of the DO is that if not k eigenvectors are targeted but rather only $k - th$ eigenvector, Davidson algorithm would still calculate some estimates of the eigenvectors 1 to $k - 1$ in order to get the $k - th$. The proposed algorithm, however, will calculate only the $k - th$ value without doing any extra job. The timings in table 6 clearly demonstrate the advantage of the algorithm over DO in the latter case, especially as k grows.

3.3 The promising ideas

We can notice that the main problem of the method's performance is IC decomposition. One way of speeding it up is to use either "negative fill", i.e. suppress the elements which are less than some minimum value. This idea was tested on the set of the artificial test matrices which were strongly diagonally dominant and showed that the speed of convergence of the

PCG doesn't decrease significantly with the fill threshold

$$\sigma = A_{11}^{11} \times 10^{-5}.$$

The disadvantage of this approach is that we don't have any clear rule for setting the σ . If we make the threshold too large, we may lose the speed of convergence, or even the convergence itself. From the other side if we set the threshold to be too small, the IC factor would remain large and the decomposition would be slow. Also, we cannot guarantee a significant savings in complexity for every case since it is quite possible that only a very few elements will be suppressed by "negative fill" approach for some matrices.

Another approach is to decompose not the full matrix but only some banded piece of it. The clear advantage of that approach is that it guarantees the low cost of the decomposition if the band is relatively narrow, while one hope to retain fast speed of the PCG convergence. However, if the most important off diagonal elements lie out of our artificial band, the IC factor loses an important information and again convergence slows down. Thus, depending on the matrix structure, one can switch between these two approaches to get the best complexity/convergence ratio.

4 Conclusion

The tests showed that assumptions about the number of required Cholesky decompositions are correct. Only in the case B1Sb was a second decomposition performed because the targeted eigenvalues were significantly different. Also the method is very promising considering the small number of matrix vector multiplies required to achieve conversion. The thorough theoretical study is still required for better understanding of the algorithm's convergence properties.

References

- [1] Davidson, E.R., "The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices." In *Journal of Computational Physics*, 17, 1975.
- [2] Stathopoulos, Andreas., Fischer. Charlotte F. "A Davidson program for Finding a Few Selected Extreme Eigenpairs of a Large, Sparse, Real, Symmetric Matrix." In *Computer Physics Communications*, pp. 268-290. vol. 79, 1994.
- [3] Fischer, Charlotte F., private communication, 1996.
- [4] Golub, G.H. and Van Loan, C.F., *Matrix Computations*, Baltimore: John Hopkins University Press, 1989.
- [5] Saad, Y., *Numerical Methods for Large Eigenvalue Problems*, Manchester: Manchester University Press, 1992.
- [6] Booten, A. and van der Vorst, H., "Cracking Large-Scale Eigenvalue Problems, Part I: Algorithms." In *Computers in Physics*, pp. 239-242, vol. 10, 1996
- [7] Saparov, Mikhail V., "Report on CS 390." Class report, Vanderbilt University, 1996
- [8] Golub, G., Ortega, James M., *Scientific Computing. An Introduction With Parallel Computing*. San Diego, California: Academic Press, Inc., 1994.
- [9] Concus, P., Golub, G.H. and Meurant, G., "Block Preconditioning For the Conjugate Gradient Method." In *SIAM Journal*, pp. 220-234, vol 6, No. 1, 1995.
- [10] Jones, Mark T. and Plassmann Paul E., "BlockSolve95 User Manual." ANL report 95/48, Argonne National Laboratory, 1995.
- [11] Greenbaum A. and Seager Mark A., "SLAP 2.0 User Guide", 1989.
- [12] Fischer C.F., "A configuration interaction program." In *Comput. Phys. Commun.*, pp. 473-493, vol. 64, 1991.
- [13] Barret. R. et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: SIAM Press, 1994.
- [14] E. Anderson et al., *LAPACK User's Guide*. Philadelphia: SIAM Press, 1992.
- [15] Wilkinson, J.H., *The Algebraic Eigenvalue Problem*. London: Oxford University Press, 1965.