Cracking large-scale eigenvalue problems, part II: Implementations

Albert Booten and Henk van der Vorst

Citation: Computers in Physics **10**, 331 (1996); doi: 10.1063/1.168590
View online: https://doi.org/10.1063/1.168590
View Table of Contents: https://aip.scitation.org/toc/cip/10/4
Published by the American Institute of Physics

CRACKING LARGE-SCALE EIGENVALUE PROBLEMS, PART II: IMPLEMENTATIONS

Albert Booten
and Henk van der Vorst

Department Editor:
William J. Thompson

This two-part article describes iterative methods for the computation of selected eigenvalues and eigenvectors of a large matrix. Part I in the previous issue reviewed the development of iterative methods, starting with the power iteration and culminating in the Jacobi–Davidson method. It showed how preconditioning can be included in the method for more efficient computation. Part II describes an implementation for the Jacobi–Davidson methods. Application of the method is illustrated by a numerical example, and Part II concludes with guidelines on where to obtain relevant free software from Netlib on the Internet.

In iterative methods for large-scale eigenvalue problems one attempts to create a low-dimensional subspace and to find the best approximation for the eigenvalue or eigenvector in that subspace. This leads to a considerably smaller eigenvalue problem that can be solved by standard direct methods. The Jacobi–Davidson (JD) method differs from the Krylov subspace methods, including the Lanczos and Arnoldi methods and the Power iteration method, in the way the subspace is expanded.

For an approximate eigenvalue θ_k and corresponding approximate eigenvector $\mathbf{u}_k$, for the generalized eigenproblem

$$\mathbf{Ax}=\lambda\mathbf{Bx}, \tag{1}$$

we define the residual $\mathbf{r}_k$ as $\mathbf{r}_k=(\mathbf{A}-\theta_k\mathbf{B})\mathbf{u}_k$.

When $\mathbf{B}=\mathbf{I}$, then for the Lanczos and Arnoldi methods the subspace is expanded with $\mathbf{r}_k$. These methods cannot be applied in a straightforward manner for $\mathbf{B}\neq\mathbf{I}$, but the JD method can. In this method the subspace is expanded with an approximation for the solution $\mathbf{z}$ from the augmented correction equation:

$$\begin{bmatrix}\mathbf{A}-\theta_k\mathbf{B} & \tilde{\mathbf{w}}\\ \tilde{\mathbf{u}}^* & 0\end{bmatrix}\begin{bmatrix}\mathbf{z}\\ \epsilon\end{bmatrix}=\begin{bmatrix}-\mathbf{r}_k\\ 0\end{bmatrix}. \tag{2}$$

As we have seen in Part I, adequate choices for the vectors $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{u}}$ are $\tilde{\mathbf{u}}=\mathbf{u}_k$ and $\tilde{\mathbf{w}}=\mathbf{Bu}_k$. We will now show how this leads to implementable algorithms.

Practical Jacobi–Davidson algorithms

In Box 1 we present a framework for practical JD algorithms. It applies to the situation in which $\mathbf{A}$ and $\mathbf{B}$ are general complex $N\times N$ matrices, with no further restrictions. For computing several eigenvalues at a time we consider a block variant, as has been frequently presented for Davidson type methods.[1-3] Suppose we want to compute l eigenpairs simultaneously. Then at each iteration we solve l correction equations approximately and subsequently add l correction vectors to the subspace. This implies that in iteration k the subspace has dimension kl. Correspondingly, the matrix $\mathbf{V}_k$ is an $N\times kl$ matrix. We now discuss some implementation aspects of the individual steps in the algorithm shown in Box 1.

Step A. Provided one has no initial information with respect to the desired eigenvectors, any choice can be made for the l starting vectors.

Step B. Apart from the matrix $\mathbf{V}_k$ it is also very convenient to store the $N\times kl$ matrices $\mathbf{W}_k^A=\mathbf{AV}_k$ and $\mathbf{W}_k^B=\mathbf{BV}_k$. Of course, in every iteration it will only be necessary to compute the last l columns of these matrices. Storage of these matrices simplifies the computation of the projected matrices $\mathbf{H}_k^A$ and $\mathbf{H}_k^B$ considerably. In every iteration only the last l columns and rows of the projected matrices are computed, since other matrix elements are available from previous iterations. In case $\mathbf{A}$ or $\mathbf{B}$ is Hermitian, only the last l columns or rows have to be calculated.

Step C. The small $kl\times kl$ projected generalized eigenproblem involves dense matrices $\mathbf{H}_k^A$ and $\mathbf{H}_k^B$ and can best be solved with the well-known QZ (or QR when $\mathbf{B}=\mathbf{I}$) algorithm.[4] The selection of the l desired eigenvalues concerns, for instance, those with the largest or smallest absolute values or those nearest to some complex value σ. In the latter case, σ is called the "shift."

Step D. In some cases it might be cheaper to compute the vectors $\mathbf{p}_{k,i}(\mathbf{q}_{k,i})$ as the matrix-vector multiplication

Albert Booten was a young and promising numerical analyst in the Center for Mathematics and Computer Science (CWI), Amsterdam. He died only a couple of days after submission of this article to CIP. The article is dedicated to his memory. Henk van der Vorst is a professor in the Department of Mathematics, Utrecht University, Utrecht, The Netherlands. He works on iterative methods, preconditioning, eigenvalue computation, and parallel computing. E-mail: vorst@math.ruu.nl

$\mathbf{p}_{k,i}=\mathbf{A}\mathbf{u}_{k,i}(\mathbf{q}_{k,i}=\mathbf{B}\mathbf{u}_{k,i})$ instead of the matrix-vector product $\mathbf{p}_{k,i}=\mathbf{W}_k^A\mathbf{s}_{k,i}(\mathbf{q}_{k,i}=\mathbf{W}_k^B\mathbf{s}_{k,i})$.

Step E. Check whether the residual norms of the desired eigenpairs are smaller than a tolerance value. If some eigenpairs have converged, it will only be necessary to compute a correction vector in step F for the remaining unconverged ones.

Step F. We suggest solving the augmented correction Eq. (2) approximately using a few iteration steps of a suitable linear system solver, possibly including a preconditioner according to Eq. (11) in Part I. Note that we set $\tilde{\mathbf{u}}$ in Eq. (2) equal to $\mathbf{u}_{k,i}$ in the algorithm. In some cases it might be convenient to retain symmetry in the coefficient matrix; then the choice $\tilde{\mathbf{u}}=\mathbf{q}_{k,i}$ should be made.

Step G. To obtain a numerically stable orthonormal basis we suggest the use of the modified Gram–Schmidt orthogonalization procedure, which can be found in almost every linear algebra text book.[4–6] In case the **B** matrix is Hermitian positive definite, the construction of a **B**-orthonormal basis is possible. This implies that $\mathbf{H}_k^B=\mathbf{I}_k$ and thus the small projected eigenproblem is a standard eigenproblem. For more details on this particular case see Ref. 7.

Step H. To limit memory requirements, it is common to restart the algorithm. Here the simplest restart strategy is suggested by retaining the current eigenvector approximations and their corresponding correction vectors in the new basis. Since this involves $2l$ vectors, we have to set the iteration counter k equal to 2. Many useful restart strategies have been suggested in the literature. For instance, in the Davidson–Liu algorithm[1] the eigenvector approximations from the previous iteration are also retained. In the algorithm we denote by m the maximum dimension of the subspace $\mathbf{V}_k$. Usually m is chosen to be a multiple of l, thus obtaining the maximum dimension in JD iteration, $k=m/l$.

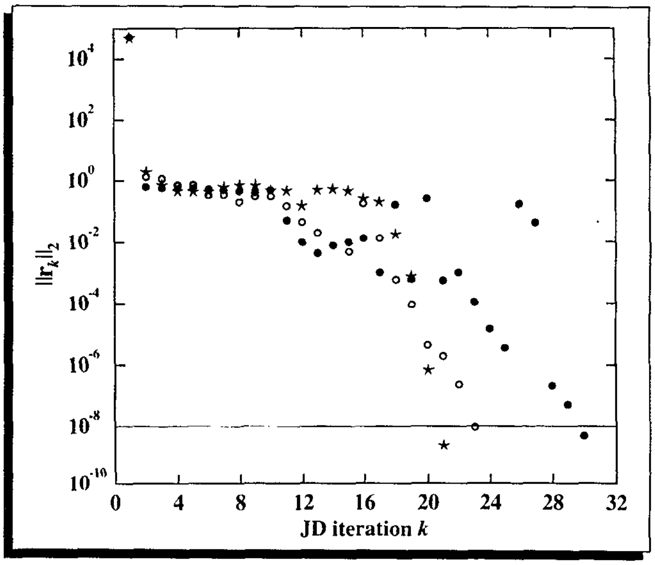

Figure 1. The residual norm in JD iteration k: correction equations solved with 20 (●), 30 (O), and 40 (⋆) GMRES iteration steps.

Box 1: Jacobi–Davidson algorithm for matrix pairs (A,B).

Step A: Choose an initial orthonormal basis matrix $\mathbf{V}_1=[\mathbf{v}_1,\ldots,\mathbf{v}_l]$. Set $k=1$.
Step B: Compute the matrices $\mathbf{W}_k^A=\mathbf{A}\mathbf{V}_k$ and $\mathbf{W}_k^B=\mathbf{B}\mathbf{V}_k$; compute the projected matrices $\mathbf{H}_k^A=\mathbf{V}_k^*\mathbf{W}_k^A$ and $\mathbf{H}_k^B=\mathbf{V}_k^*\mathbf{W}_k^B$.
Step C: Solve the small projected generalized eigenproblem $\mathbf{H}_k^A\mathbf{s}_{k,i}=\theta_{k,i}\mathbf{H}_k^B\mathbf{s}_{k,i}$ and select the l desired eigenpairs $[(\theta_{k,i},s_{k,i});\ i=1,\ldots l]$.
Step D: For $i=1,\ldots,l$ compute: the Ritz vectors $\mathbf{u}_{k,i}=\mathbf{V}_k\mathbf{s}_{k,i}$; the vectors $\mathbf{p}_{k,i}=\mathbf{W}_k^A\mathbf{s}_{k,i}$ and $\mathbf{q}_{k,i}=\mathbf{W}_k^B\mathbf{s}_{k,i}$; the residual vectors $\mathbf{r}_{k,i}=\mathbf{p}_{k,i}-\theta_{k,i}\mathbf{q}_{k,i}$.
Step E: Test for convergence. If satisfied, stop.
Step F: For $i=1,\ldots,l$ compute an approximate solution $\tilde{\mathbf{z}}_{k,i}$ of the augmented correction equation:

$$\begin{bmatrix}\mathbf{A}-\theta_{k,i}\mathbf{B} & \mathbf{q}_{k,i}\\ \mathbf{u}_{k,i}^* & 0\end{bmatrix}\begin{bmatrix}\mathbf{z}_{k,i}\\ \epsilon_{k,i}\end{bmatrix}=\begin{bmatrix}-\mathbf{r}_{k,i}\\ 0\end{bmatrix}.$$

Step G: If $\dim(\mathbf{V}_k)\leqslant m-l$, then orthonormalize $(\mathbf{V}_k,\tilde{\mathbf{z}}_{k,1},\ldots,\tilde{\mathbf{z}}_{k,l})$ into $\mathbf{V}_{k+1}$ via modified Gram–Schmidt; increase k by 1 and return to step *B*.
Step H: Restart: orthonormalize $(\mathbf{u}_{k,1},\ldots,\mathbf{u}_{k,l},\tilde{\mathbf{z}}_{k,1},\ldots,\tilde{\mathbf{z}}_{k,l})$ into $\mathbf{V}_2$ via modified Gram–Schmidt; set $k=2$ and return to step B.

It should be stressed that the JD method for generalized eigenvalue problems does not require the inversion of a matrix. This may be a big advantage over other methods such as the Lanczos and Arnoldi methods.[5] These methods transform the generalized eigenproblem $\mathbf{Ax}=\lambda\mathbf{Bx}$ into an equivalent standard eigenproblem $\tilde{\mathbf{A}}\mathbf{x}=\lambda\mathbf{x}$ by inverting a matrix. For instance, the generalized problem $\mathbf{Ax}=\lambda\mathbf{Bx}$ is equivalent to the standard problem for the matrix $\mathbf{B}^{-1}\mathbf{A}$. The matrix $\mathbf{B}^{-1}$ is not computed in practice, but a factorization of **B** into the product of a lower triangular **L** and upper triangular matrix **U**. A matrix-vector multiplication of the type $\mathbf{y}=\mathbf{B}^{-1}\mathbf{z}$ is then computed by solving $\mathbf{By}=\mathbf{z}$, or, which is similar, $\mathbf{LUy}=\mathbf{z}$. The latter is done by a forward and back substitution (see Ref. 4). This procedure for solving $\mathbf{By}=\mathbf{z}$ is the well-known Gaussian elimination method. The computation of the **L** and **U** factors can be very expensive. Furthermore, storage of the factors **L** and **U** may require an amount of memory that greatly exceeds the memory necessary to store matrices **A** and **B** themselves.

Numerical experiments

The Jacobi–Davidson method has recently been applied successfully to symmetric standard eigenvalue problems arising in quantum chemistry[8] ($N\approx50{,}000$), quadratic eigenvalue problems arising in acoustics[9] ($N>100000$), and non-Hermitian generalized eigenproblems ($N<1000$) from magneto hydrodynamics (MHDs) spectroscopy.[10] We now give some results for a fairly large MHD test problem ($N=4800$). The **A** matrix is non-Hermitian; the **B** matrix is Hermitian positive definite. Both matrices have a block tridiagonal structure. The goal is to compute the eigenvalue closest to the shift $\sigma=-0.3+0.6i$, which is deep in the interior of the spectrum. Other eigenvalue solvers usually have great difficulty in finding interior eigenvalues if one wants to avoid inversion of any of the matrices involved, although Ruhe[11] has proposed a variant of the Lanczos method for interior eigenvalues: the rational Lanczos method.

In the algorithm presented in Box 1 we set $l=1$ and the maximum subspace dimension $m=40$. For the starting vector $\mathbf{v}_1$ we chose the vector with all components equal to 1 divided by the normalization factor. In every JD iteration the augmented correction Eq. (2) is solved to some approximation with a fixed number of iteration steps of the GMRES[12] algorithm. The correction equations are preconditioned according to Eq. (11) in Part I, with $\mathbf{K}$ an incomplete $\mathbf{LU}$ factorization of $\mathbf{A}-\sigma\mathbf{B}$, that is, in the $\mathbf{LU}$-decomposition process certain matrix elements are set to zero (dropped) during the process. For further references on this see Ref. 13. For the experiments shown here, we follow a procedure suggested by Saad.[14] We point out that the preconditioner is computed only once and that it gets more effective as the eigenvalue approximation θ_k comes closer to σ.

The JD iteration process is stopped if the residual norm $\|\mathbf{r}_k\|_2$ is smaller then 10^{-8}. The convergence to the desired eigenvalue $-0.2855868...+0.5470253...i$ is plotted in Fig. 1 for three runs of the algorithm: solving the linear systems with 20, 30, and 40 GMRES iteration steps, respectively. As expected, the better we solve the linear systems, which corresponds to more GMRES steps, the faster convergence is reached. For 40 GMRES steps the JD process terminates in only 21 iterations, for 30 GMRES steps it terminates in 23 iterations, and for 20 GMRES steps it terminates in 30 iterations.

However, if we look at the corresponding CPU times obtained on a Cray C98, the speed of convergence is reversed: for 40 GMRES steps convergence is obtained in 469 s, for 30 GMRES steps in 372 s, and for 20 GMRES steps in only 322 s. In Fig. 1 we see that in the beginning of the JD process virtually the same progress is made for the eigenpair approximation, whether we solve the systems with 20, 30, or 40 GMRES steps. This indicates a stagnation in the iterative method for solving the systems. Only in the final iterations does it seem to pay off to solve the systems more accurately, resulting in asymptotically very fast convergence for 40 GMRES steps. Therefore we can improve these results by building in a "switch" value: we start by solving the correction equations with 20 GMRES steps, but as soon as the residual norm is smaller than 10^{-2} we switch to 40 GMRES steps. The results are shown in Fig. 2. Convergence is obtained in only 19 JD iterations, requiring only 277 s of CPU time.

We also tried to reproduce the eigenvalue by using the generalized Davidson (GD) method (see Part I), using a similar incomplete factorization for the GD correction equation. All parameters were set to the same values as for the JD method. However, all our attempts with the GD method failed for this test problem.

Software for large-scale eigenvalue problems

It is relatively easy to code the JD method as given in Box 1. The two main ingredients of a JD code are a dense-matrix eigenvalue solver (step C) and a sparse-matrix linear system solver (step F). Appropriate Fortran routines are available from Netlib.[15] Netlib is accessible via the World Wide Web: http://www.netlib.org/.

For the eigenvalue solver we suggest software from the linear algebra package LAPACK.[16] For eigenproblems it contains driver routines that compute the entire spectrum of a matrix or matrix pair. Routines are available for Hermitian, non-Hermitian, standard, and generalized eigenproblems. The LAPACK user's guide is available online from the Web page http://www.netlib.org/lapack/lug/lapack_lug.html.

Efficient Fortran (and C) codes for the iterative solution of linear systems have recently become available via the Web page http://www.netlib.org/templates/Templates.html. Routines are included for all popular linear system solvers such as the conjugate gradient method; see, for example, Ref. 4 for symmetric systems or the GMRES method[12] for nonsymmetric systems. A description of all the methods is given in Ref. 13; the inside cover of this book shows a flow chart with suggestions for the selection of a suitable iterative method for a given coefficient matrix. The book is available online: http://www.netlib.org/templates/templates.ps.

For each of the linear system solvers, the user needs to supply an additional subroutine that computes a matrix-vector multiplication. Such routines depend on the way the user has organized the nonzero matrix elements in the program. The book discusses a variety of sparse-matrix storage formats and suggests suitable implementations of the matrix-vector product. For many problems it is necessary to combine the iterative solver with a suitable preconditioner.

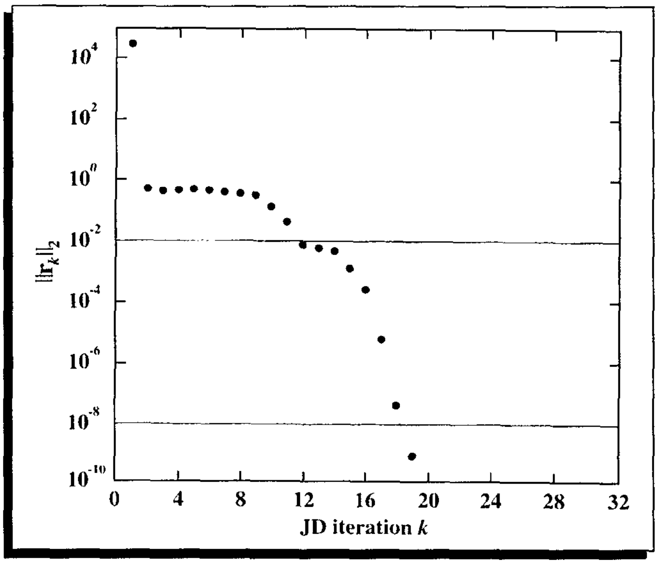

Figure 2. The residual norm in JD iteration k: as soon as $\|r_k\|_2<10^{-2}$, switch from 20 to 40 GMRES iteration steps.

The preconditioning routine is also user supplied. Many possibilities for suitable preconditioners are discussed in Ref. 13. Readers without access to the World Wide Web can obtain all the information from Netlib[15] by sending electronic mail to the address netlib@ornl.gov. The subject line "Subject: send index from templates" returns the index of the appropriate software library. Individual routines and, for instance, the ps-file of Ref. 13 can be obtained by giving subject lines of the following type: "Subject: send templates.ps from templates."

References

1. E. R. Davidson, Comput. Phys. **7**, 519 (1993).
2. M. Crouzeix, B. Philippe, and M. Sadkane, SIAM J. Sci. Comp. **15**, 62 (1994).
3. M. Sadkane, Numer. Math. **64**, 195 (1993).
4. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. (The Johns Hopkins University Press, Baltimore, 1989).
5. Y. Saad, *Numerical Methods for Large Eigenvalue Problems* (Manchester University Press, Manchester, 1992).
6. B. N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice–Hall, Englewood Cliffs, New Jersey, 1980).
7. G. L. G. Sleijpen, J. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst, Preprint No. 923, Department of Mathematics, Utrecht University, Utrecht, The Netherlands, September 1995.
8. H. J. J. van Dam, J. H. van Lenthe, G. L. G. Sleijpen, and H. A. van der Vorst, J. Comput. Chem. **17**, 267 (1996).
9. M. B. van Gijzen and F. A. Raeven, Preprint No. 924, Department of Mathematics, Utrecht University, Utrecht, The Netherlands, September 1995.
10. A. Booten, D. Fokkema, G. Sleijpen, and H. van der Vorst, CWI Report NM-R9514, CWI, Amsterdam, The Netherlands, August 1995.
11. A. Ruhe, Linear Algebra Appl. **58**, 391 (1984).
12. Y. Saad and M. H. Schultz, SIAM J. Sci. Stat. Comp. **7**, 856 (1986).
13. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, Philadelphia, 1994).
14. Y. Saad, Num. Lin. Alg. Appl. **1**, 387 (1994).
15. J. J. Dongarra and E. Grosse, Comm. ACM **30**, 403 (1987).
16. 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 Users' Guide, Release 2.0* (SIAM, Philadelphia, 1995).