Parallel Solvers for Large Eigenvalue Problems Originating from Maxwell's Equations

Peter Arbenz[1] and Roman Geus[1]

Swiss Federal Institute of Technology (ETH), Institute of Scientific Computing,
CH-8092 Zurich,
{arbenz,geus}@inf.ethz.ch

Abstract. We present experiments with two new solvers for large sparse symmetric matrix eigenvalue problems: (1) the implicitly restarted Lanczos algorithm and (2) the Jacobi-Davidson algorithm. The eigenvalue problems originate from in the computation of a few of the lowest frequencies of standing electromagnetic waves in cavities that have been discretized by the finite element method. The experiments have been conducted on up to 12 processors of an HP Exemplar X-Class multiprocessor computer.

1 Introduction

Most particle accelerators use standing waves in cavities to produce the high voltage RF (radio frequency) fields required for the acceleration of the particles. The mathematical model for these high frequency electromagnetic fields is the eigenvalue problem solving Maxwell's equations in a bounded volume. Usually, the eigenfield corresponding to the fundamental mode of the cavity is used as the accelerating field. Due to higher harmonic components contained in the RF power fed into the cavity, and, through interactions between the accelerated particles and the electromagnetic field, an excitation of higher order modes can occur. The RF engineer designing such an accelerating cavity therefore needs a tool to compute the fundamental and about ten to twenty of the following eigenfrequencies together with the corresponding electomagnetic eigenfields.

After separation of variables depending on space and time and after elimination of the magnetic field terms the variational form of the eigenvalue problem for the electric field intensity is given by [1]

$$\begin{aligned} &\textit{Find } (\lambda, \mathbf{u}) \in \mathbb{R} \times W_0 \textit{ such that } \mathbf{u} \neq \mathbf{0} \textit{ and} \\ &(\mathbf{curl}\,\mathbf{u}, \mathbf{curl}\,\mathbf{v}) = \lambda(\mathbf{u}, \mathbf{v}), \qquad \forall \mathbf{v} \in W_0. \end{aligned} \tag{1}$$

Let $L^2(\Omega)$ be the Hilbert space of square-integrable functions over the 3-dimensional domain Ω (the cavity) with inner product $(\mathbf{u}, \mathbf{v}) = \int_\Omega \mathbf{u}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x})\, d\mathbf{x}$ and norm $\|\mathbf{u}\|_{0,\Omega} := (\mathbf{u}, \mathbf{u})^{1/2}$. In (1), $W_0 := \{\mathbf{v} \in W \mid \operatorname{div} \mathbf{v} = 0\}$ with

$$W := \{\mathbf{v} \in L^2(\Omega)^3 \mid \mathbf{curl}\,\mathbf{v} \in L^2(\Omega)^3\ ,\ \operatorname{div} \mathbf{v} \in L^2(\Omega)\ ,\ \mathbf{n} \times \mathbf{v} = 0 \text{ on } \partial\Omega\}.$$

The difficulty with (1) stems from the condition $\operatorname{div} \mathbf{v} = 0$ as it is hard to find divergence-free finite elements. Therefore, ways have been looked for to get around the condition $\operatorname{div} \mathbf{v} = 0$. In this process care has to be taken in order not to introduce so-called *spurious modes*, i.e. eigenmodes that have no physical meaning [8]. We considered two approaches free of spurious modes, a penalty method and an approach based on Lagrange multipliers. In the *penalty method* approach (1) is replaced by [9, 11]

$$\begin{aligned} &\textit{For fixed } s > 0, \textit{ find } (\lambda, \mathbf{u}) \in \mathbb{R} \times W \textit{ such that } \mathbf{u} \neq \mathbf{0} \textit{ and} \\ &(\mathbf{curl}\, \mathbf{u}, \mathbf{curl}\, \mathbf{v}) + s\,(\operatorname{div} \mathbf{u}, \operatorname{div} \mathbf{v}) = \lambda(\mathbf{u}, \mathbf{v}), \qquad \forall \mathbf{v} \in W. \end{aligned} \tag{2}$$

Here, s is a positive, usually small parameter. The eigenmodes $\mathbf{u}(\mathbf{x})$ of (2) corresponding to eigenvalues $\lambda < \mu_1 s$ are eigenmodes of (1). μ_1 is the smallest eigenvalue of the negative Laplace operator $-\Delta$ on Ω.

When discretized by ordinary nodal-based finite elements, (2) leads to matrix eigenvalue problems of the form

$$A\mathbf{x} = \lambda M \mathbf{x}. \tag{3}$$

For positive s, both A and M are symmetric positive definite sparse n-by-n matrices.

In the *mixed formulation* the divergence-free condition is enforced by means of Lagrange multipliers [9].

$$\begin{aligned} &\textit{Find } (\lambda, \mathbf{u}, p) \in \mathbb{R} \times H_0(\mathbf{curl}; \Omega) \times H_0^1(\Omega) \textit{ such that } \mathbf{u} \neq \mathbf{0} \textit{ and} \\ &\text{(a) } (\mathbf{curl}\, \mathbf{u}, \mathbf{curl}\, \boldsymbol{\Psi}) + (\mathbf{grad}\, p, \boldsymbol{\Psi}) = \lambda(\mathbf{u}, \boldsymbol{\Psi}), \quad \forall \boldsymbol{\Psi} \in H_0(\mathbf{curl}; \Omega) \\ &\text{(b) } (\mathbf{u}, \mathbf{grad}\, q) = 0, \qquad\qquad\qquad\quad \forall q \in H_0^1(\Omega) \end{aligned} \tag{4}$$

Here, $H_0(\mathbf{curl}; \Omega) := \{\mathbf{v} \in L^2(\Omega)^3 \mid \mathbf{curl}\, \mathbf{v} \in L^2(\Omega)^3,\ \mathbf{n} \times \mathbf{v} = 0 \text{ on } \partial\Omega\}$. The finite element discretization of (4) yields a matrix eigenvalue problem of the form

$$\begin{bmatrix} A & C \\ C^T & O \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \lambda \begin{bmatrix} M & O \\ O & O \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}. \tag{5}$$

where A and M are n-by-n and C is n-by-m. M is positive definite, A is only positive semidefinite. It turns out that (5) becomes most convenient to handle if the finite elements for the vector fields are chosen to be edge elements as proposed by Nédélec [12, 8] and the Lagrange multipliers are represented by nodal-based finite elements of the same degree. Then, the columns of $M^{-1}C$ form a basis for the nullspace of A and, in principle, it suffices to compute the eigenvalues of an eigenproblem formally equal to (3) but with A and M from (5). To get rid of the high-dimensional eigenspace associated with the eigenvalue zero Bespalov [4] proposed to replace (5) by

$$\tilde{A}\mathbf{x} = \lambda M \mathbf{x}, \qquad \tilde{A} = A + CHC^T, \tag{6}$$

where H is a positive definite matrix chosen such that the zero eigenvalues are shifted to the right of the desired eigenvalues and do not disturb the computations.

In this note we consider solvers for $A\mathbf{x} = \lambda M \mathbf{x}$ with A and M from the penalty approach (3) as well as from the mixed element approach (6).

2 Algorithms

In this section we briefly survey the numerical methods that we will apply to the model problem which is a cavity of the form of a rectangular box, $\Omega = (0,a) \times (0,b) \times (0,c)$. We investigate two algorithms for computing a few of the smallest (positive) eigenvalues of the matrix eigenvalue problems originating from both the penalty method and the mixed method.

For computing a few, say p, eigenvalues of a sparse matrix eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad A = A^T, \quad M = M^T > 0, \tag{7}$$

closest to a number τ it is advisable to make a so-called *shift-and-invert* approach and apply a spectral transformation with a shift σ close to τ and solve [7]

$$(A - \sigma M)^{-1} M\mathbf{x} = \mu\mathbf{x}, \qquad \mu = \frac{1}{\lambda - \sigma}. \tag{8}$$

instead of solving (7). Notice that $(A - \sigma M)^{-1}M$ is M-symmetric, i.e., it is symmetric with respect to the inner product $\mathbf{x}^T M\mathbf{y}$. The spectral transformation leaves the eigenvectors unchanged. The eigenvalues of (7) close to the *shift* σ become the largest absolute of (8). They are relatively well-separated which improves the speed of convergence. The cost of the improved convergence rate is the need to solve (at least approximately) systems of equations with the matrix $A - \sigma M$. In all algorithms the shift σ was chosen to be $48 < \lambda_1 \approx 48.4773$.

1. The Implicitly Restarted Lanczos algorithm (IRL). Because of the large memory consumption of the Lanczos algorithm it is often impossible to proceed until convergence. It is then necessary to *restart* the iterative process in some way with as little loss of information as possible. An elegant way to restart has been proposed by Sorensen for the Arnoldi algorithm [16], see [5] for the symmetric Lanczos case. Software is publicly available in ARPACK [10]. The algorithm is based on the spectral transformation Lanczos algorithm. The iteration process is executed until $j = p + k$, where k is some positive integer, often $k = p$. Complete reorthogonalization is done for stability reasons. This is possible since by assumption $p + k$ is not big.

In the restarting phase a clever application of the QR algorithm [14] reduces the dimension of the search space to p in such a way that the p new orthogonal basis vectors still form a Krylov sequence [16,5]. This allows the Lanczos phase to be resumed smoothly.

Here, we solved the symmetric indefinite system of equations $(A - \sigma M)\mathbf{x} = \mathbf{y}$ iteratively by SYMMLQ [13,3]. The accuracy of the solution of the linear system has to be at least as high as the desired accuracy in the eigenvalue calculation in order that the coefficients of the Lanczos three-term recurrence are reasonably accurate [10]. We experimented with various preconditioners. None of them was satisfactory. We obtained the best results with diagonal preconditioning. In our implementation we chose $p = k = 15$. Besides the storage for the matrices A and M, IRL requires space for two $n \times (p + k)$ arrays.

2. The Jacobi-Davidson algorithm (JDQR). In the Jacobi-Davidson algorithm the eigenpairs of $A\mathbf{x} = \lambda M\mathbf{x}$ are computed one by one. As with the implicitly restarted Lanczos procedure the search space V_j is expanded up to a certain dimension, say $j = j_{\max}$. The basis of this space is constructed to be M-orthogonal but there is no three-term recurrence relation. In the expansion phase the search space is extended by a vector $\mathbf{v}$ orthogonal to the current eigenpair approximation $(\tilde{\lambda}, \tilde{\mathbf{q}})$ by means of equation [15]

$$(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)(A - \tilde{\lambda}M)\mathbf{v} = -(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{r}, \qquad (I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{v} = \mathbf{v}. \quad (9)$$

(If eigenvectors have already been computed, this process is executed in the space M-orthogonal to them.) The solution of (9) is only needed approximately. Therefore, it can be solved iteratively. In [6] the authors propose to use a preconditioner of the form

$$(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)K(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M) \quad (10)$$

where K is a good and easily invertible approximation of $A - \tilde{\lambda}M$. They also give a generalized inverse of the matrix in (10). We solved the systems by the conjugate gradient squared (CGS) method [3] with K equal to the diagonal of $A - \tilde{\lambda}M$. The next approximate eigenvalue $\tilde{\lambda}$ and corresponding Ritz vector $\tilde{\mathbf{q}}$ are obtained by a Ritz step for the subspace V_{j+1}. If the search space has dimension $j_{\max}$ it is shrunk to dimension $j_{\min}$ by selecting only those $j_{\min}$ Ritz vectors corresponding to Ritz values closest to the target value $\tau = 48$. In our experiments we set $j_{\min} = p = 15$ and $j_{\max} = 2p$ as suggested in [10].

Thus, besides the storage for the matrices A and M, memory space is needed for a $n \times j_{\max}$ array and for three $n \times p$ arrays.

3 Numerical Experiments

In this section we compare IRL and JDQR for computing the 15 smallest eigenvalues of (1) with $\Omega = (0, a) \times (0, b) \times (0, c)$ where $a = 0.8421$, $b = 0.5344$, $c = 0.2187$. We apply penalty as well as mixed methods to this model problem whose eigenvalues can be computed analytically [1]. The computational results have been obtained with the HP Exemplar X-class system at the ETH Zurich.

1. Sequential results. In Figs. 1 and 2 execution times for computing the 15 lowest eigenvalues and associated eigenvectors vs. the accuracy of the computed solutions relative to the analytic solution are plotted for different mesh sizes. The largest problems sizes were $n = 65'125$ and $n = 45'226$ for the linear and quadratic edge elements, and $n = 28'278$ and $n = 118'539$ for the linear and quadratic node elements. For all experiments we used a tolerance of 10^{-8} in the stopping criterion of the eigensolvers. Loosely speaking, this means that the eigenvalues are computed to at least 8 significant digits. The execution times comprise the solution time of the eigensolver but not the time for building the matrices. For each finite element type (linear/quadratic, node element/edge element) the performance of the two algorithms is shown. Fig. 1 shows the results

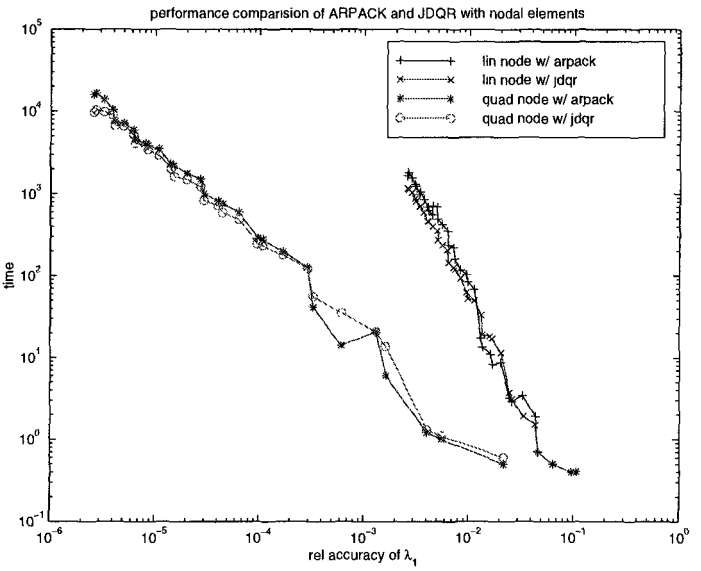

Fig. 1. Comparison of IRL and JDQR with node elements: Accuracy of the computed λ_1 relative to the analytic solution vs. computation time

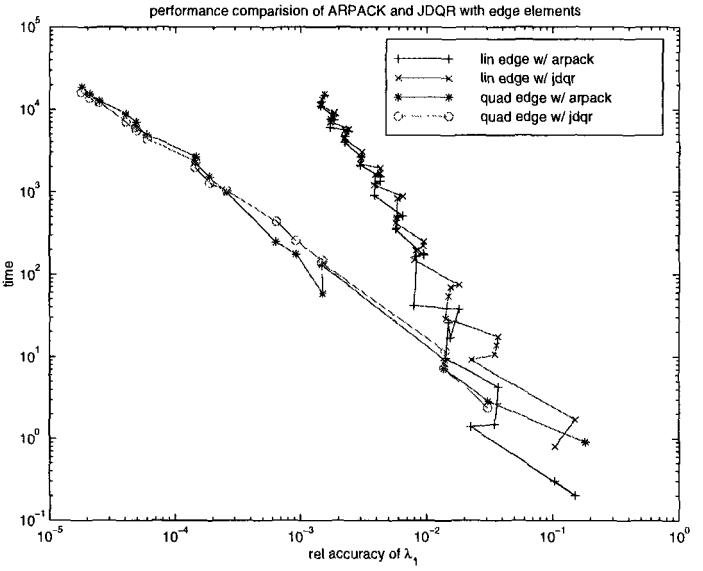

Fig. 2. Comparison of IRL and JDQR with edge elements: Accuracy of the computed λ_1 relative to the analytic solution vs. computation time

that we obtained with linear and quadratic *node* elements for λ_1. The higher eigenvalues behave similarly but are of course less accurate. Fig. 2 shows the results for the linear and quadratic *edge* elements.

For edge elements the convergence is improved by introducing the matrix C as given in (6). Experiments to that end are reported in [2]. H in (6) was heuristically chosen to be αI with $\alpha = 100/h$, where h is the mesh width.

The comparison of *linear* with *quadratic* element types reveals immediately the inferiority of the former. They give much lower accuracy for a given computation time. The node elements in turn are to be preferred to the edge elements, at least in this simplified model problem. For a given computational effort, the eigenvalues obtained with the node elements are about an order of magnitude more accurate than those obtained with the edge elements. The situation is not so evident with the linear elements.

With the quadratic elements and linear node elements JDQR is consistently faster than IRL by about 10 to 30% for the large problem sizes. For linear edge elements JDQR still is 10 to 20% ahead of IRL for most of the larger problem sizes. For the smaller problems the situation is not so clear.

We now discuss the behavior of the two algorithms in the case of the quadratic edge elements where the problem size grows from $n = 1408$ to 45226. (In the latter case A and M have each about 1'730'000 nonzero elements.) With both algorithms the number of outer iteration steps varies only little. It is ~ 65 with IRL and ~ 210 with JDQR. So, the number of restarts does not depend on the problem size. Each outer iteration step requires the solution of one system of equations. We used the solver SYMMLQ with IRL and CGS for JDQR. One (inner) iteration step of CGS counts for about two iteration steps of CGS. We applied diagonal preconditioning in all cases. The superiority of JDQR over IRL for large problem sizes can be explained by the number of inner iteration steps. The average number of inner iteration steps per outer iteration step grows from 64 to 91 with JDQR but from 423 to 1140 with IRL. In ARPACK each system of equations is solved to high accuracy. In JDQR the accuracy requirement is not so stringent and actually varies from (outer) step to step, cf. §2.2. This explains the higher iteration numbers for IRL. The projections in (9) improve the condition number of the system matrix. Further, the shift σ is updated in JDQR while it stays constant with ARPACK. These may be the reasons for the less pronounced increase of the number of inner iteration steps with JDQR. Notice that the projections made in each iteration step account for less than 10% of the execution time of JDQR. Similar observations can be made with the other element types.

2. Parallel results. The parallel experiments were carried out on the 32 processor HP Exemplar X-class system at ETH Zurich. This shared-memory CC-NUMA machine consists of two hypernodes with 16 HP PA-8000 processors and 4 GBytes memory each. The processors and memory-banks within a hypernode are connected through a crossbar-switch with a bandwidth of 960 MBytes/s per port. The hypernodes themselves are connected by a slower network with a ring

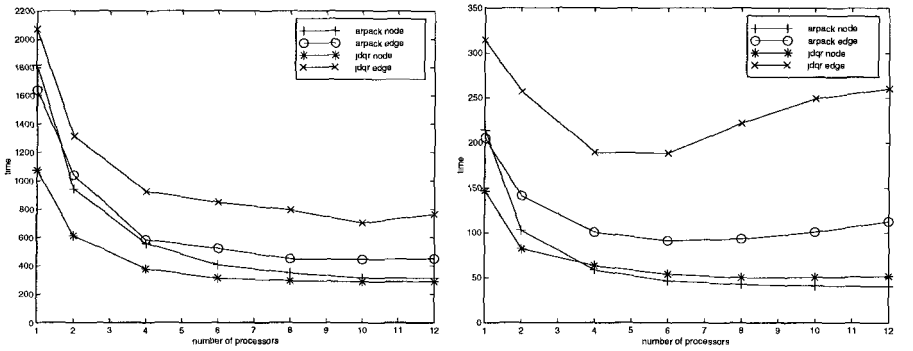

Fig. 3. Execution times (a) for the large eigenproblem ($n = 22323$ for node, $n = 5798$ for edge elements) and (b) for the small eigenproblem ($n = 4981$ and $n = 1408$, respectively)

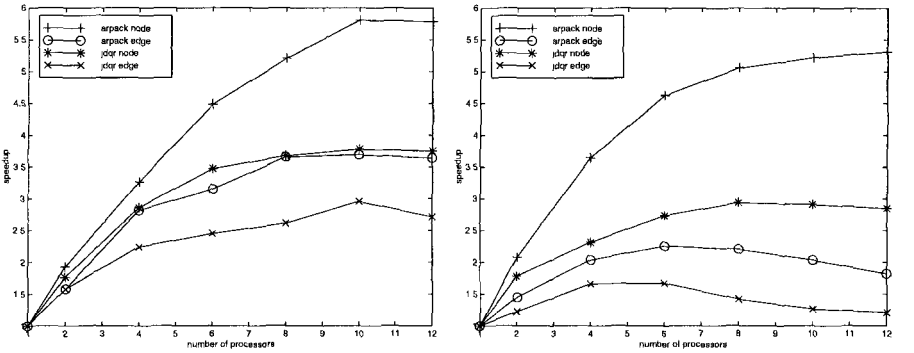

Fig. 4. Speedups (a) for the large eigenproblem and (b) for the small eigenproblem

topology. The HP PA-8000 processors have a clock rating of 180 MHz and a peak performance of 720 MFLOPS.

We carried out the parallel experiments for IRL (ARPACK) and JDQR using both quadratic node and quadratic edge elements and two different problem sizes. Linear elements are not considered because of the inferior results in the sequential case. For both edge and node elements we chose two problem sizes: the larger problems require about 1700 seconds one processor, whereas the smaller problems require only about 220 seconds. For our experiments we were allowed to use up to 12 processors.

With ARPACK about 95% of the sequential computation time is spent in the inner loops forming sparse matrix-vector products with the matrices A, M and C. JDQR spends about 90% for this task. We therefore parallelized the sparse matrix-vector product using the so called HP Exemplar advanced shared-memory programming technique, that is directive-based.

The matrix C and the strictly lower triangles of A and M are stored in the compressed sparse row format [3]. The diagonals of A and M are stored separately. We parallelized the outermost loops with the `loop_parallel` directives. The necessary privatization of variables was done manually by means of the `loop_private` directive. For the product with C and the lower triangular parts of A and M no special considerations were necesary. For the product with C^T and the upper triangular parts of A and M the processors store their "local" result into distinct vectors, that are accumulated into the global result vector afterwards. To get a well-balanced work load we distributed the triangular matrices in block-cyclic fashion over the processors.

In Fig. 3 the execution times for both problem sizes are plotted. In Fig. 4 the corresponding speedups are found. The best speedups are reached with 10 processors for the large problems. ARPACK's IRL gives speedups of 5.8 for node elements and of 3.7 for edge elements. With JDQR we get 3.8 and 3.0, respectively. These numbers are lower for the small problem size.

The reason for the better speedups with ARPACK is, that the implicit classical Gram-Schmidt orthogonalization which consumes between 5 and 10% of the sequential execution time in JDQR doesn't scale well on the HP-Exemplar. The parallelized BLAS routine `DGEMV` in the HP MLIB library shows no speedup even though the matrices have more than 100'000 nonzero elements! We are currently resolving this issue with HP. Using a reasonably parallelizing `DGEMV` routine we expect JDQR to scale as well as ARPACK.

Fig. 4 further shows that in our experiments node elements scale better than edge elements. Since we chose the problem sizes such that both edge and node elements require about the same computation time on one processor, and the linear systems arising from node elements are better conditioned, the matrices originating from edge elements are smaller. Matrices A and M stemming from edge elements together have about 7 times fewer non-zero elements than in the nodal case. So, the relative parallelization overhead is bigger for edge elements. For IRL the situation is improved because instead of A we store the shifted matrix $A - \sigma M$, which has about twice as many non-zero elements.

Our parallel experiments show the limitations of directive-based shared-memory programming. Before and after each parallel section (e.g. parallel loops) of the code, the compiler inserts global synchronization operations to ensure correct execution. However, in most cases, these global synchronization operations are unnecessary. That is why our implementation doesn't scale well. In order to remove unnecessary synchronization points, a different programming paradigm, such as message-passing or low-level shared-memory programming, has to be employed.

Our parallel results depend very much on the computer architecture, the parallel libraries and the programming paradigm we used. They can therefore not be easily generalized. Nevertheless, our experiments prove that reasonable parallel performance can be obtained on small processor numbers with only modest programming effort using directive-based shared-memory programming on the HP Exemplar.

References

1. St. Adam, P. Arbenz, and R. Geus, *Eigenvalue solvers for electromagnetic fields in cavities*, Tech. Report 275, ETH Zürich, Computer Science Department, October 1997, (Available at URL `http://www.inf.ethz.ch/publications/tr.html`).
2. P. Arbenz and R. Geus, *Eigenvalue solvers for electromagnetic fields in cavities*, FORTWIHR International Conference on High Performance Scientific and Engineering Computing (F. Durst et al., ed.), Springer-Verlag, 1998, (Lecture Notes in Computational Science and Engineering).
3. R. Barret, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, Ch. Romine, and H. van der Vorst, *Templates for the solution of linear systems: Building blocks for iterative methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994, (Available from Netlib at URL `http://www.netlib.org/templates/index.html`).
4. A. N. Bespalov, *Finite element method for the eigenmode problem of a RF cavity resonator*, Soviet Journal of Numerical Analysis and Mathematical Modelling **3** (1988), 163–178.
5. D. Calvetti, L. Reichel, and D. C. Sorensen, *An implicitely restarted Lanczos method for large symmetric eigenvalue problems*, Electronic Transmissions on Numerical Analysis **2** (1994), 1–21.
6. D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils*, Preprint 941, revised version, Utrecht University, Department of Mathematics, Utrecht, The Netherlands, January 1997.
7. R. Grimes, J. G. Lewis, and H. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM J. Matrix Anal. Appl. **15** (1994), 228–272.
8. J. Jin, *The finite element method in electromagnetics*, Wiley, New York, 1993.
9. F. Kikuchi, *Mixed and penalty formulations for finite element analysis of an eigenvalue problem in electromagnetism*, Computer Methods in Applied Mechanics and Engineering **64** (1987), 509–521.
10. R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide: Solution of large scale eigenvalue problems by implicitely restarted Arnoldi methods*, Department of Mathematical Sciences, Rice University, Houston TX, October 1997, (Available at URL `http://www.caam.rice.edu/software/ARPACK/index.html`).
11. R. Leis, *Zur Theorie elektromagnetischer Schwingungen in anisotropen Medien*, Mathematische Zeitschrift **106** (1968), 213–224.
12. J. C. Nédélec, *Mixed finite elements in* $\mathbb{R}^3$, Numerische Mathematik **35** (1980), 315–341.
13. C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal. **12** (1975), 617–629.
14. B. N. Parlett, *The symmetric eigenvalue problem*, Prentice Hall, Englewood Cliffs, NJ, 1980.
15. G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst, *Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems*, BIT **36** (1996), 595–633.
16. D. Sorensen, *Implicite application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl. **13** (1992), 357–385.