Open Problems in Numerical Linear Algebra

J.W. Demmel

CRPC-TR92421
1992

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

This work was supported in part by the NSF and DARPA.

Open Problems in Numerical Linear Algebra

J. W. Demmel*
Computer Science Division and Mathematics Department
University of California
Berkeley, CA 94720

Abstract

The original goal of the LAPACK project was to design and implement a portable linear algebra library that would be very efficient on high-performance machines. During the project it became apparent we could also significantly improve the accuracy of many standard algorithms in linear algebra, with little or no sacrifice of speed. This work has led to new perturbation theory, new algorithms and new error analyses for many problems, as well as many still unsolved problems. In this paper we survey some of these new results, and discuss open problems in four related areas: high accuracy algorithms, parallel algorithms, the complexity of condition estimation, and exploiting IEEE standard floating point arithmetic.

1 Introduction

The University of Tennessee, the Courant Institute for Mathematical Sciences, the Numerical Algorithms Group, Ltd., Rice University, Argonne National Laboratory, Oak Ridge National Laboratory, and the University of California at Berkeley have developed a transportable linear algebra library in Fortran 77. The library is intended to provide a uniform set of subroutines to solve the most common linear algebra problems and to run efficiently on a wide range of high-performance computers.

The LAPACK library (shorthand for Linear Algebra Package) provides routines for solving systems of simultaneous linear equations, least-squares solutions of overdetermined systems of equations, and eigenvalue problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur) are also provided, as well as related computations such as reordering of the factorizations. Dense and banded matrices are provided for, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices. The software is in the public domain and is available from netlib [35]. For a more complete survey of LAPACK, see [2, 1].

The original goal of LAPACK was simply to be faster than its predecessors EISPACK [81, 44] and LINPACK [29], which run inefficiently on machines with hierarchical memories.

*This paper was written while the author was visiting Clemson University and the Institute for Mathematics and its Applications at the University of Minnesota. The author also acknowledges the support of NSF grants ASC-9005933 and DCR-8552474, and DARPA grant DAAL03-91-C-0047 via a subcontract from the University of Tennessee.

In the course of the project, it was also discovered that many standard problems could also be solved much more accurately than before. This was done by discovering algorithms whose backward error was significantly smaller than that of conventional algorithms [4, 26, 16, 8, 27]. This permitted us, for example to compute singular values of bidiagonal matrices to high relative accuracy no matter how tiny they are [26]; in contrast, the standard algorithm [50] may compute the tiny singular values without any relative accuracy at all.

As a result of these successes, our vision of what linear algebra software should provide has changed considerably: our expectations of accuracy have risen. In addition, there are a great many new problems to solve (and new techniques to try) in order to expand these results to more algorithms. We will discuss these open problems in section 2 below.

On the other hand, there remain some problems for which no highly parallel algorithms exist that are numerically stable even in the conventional sense. For some problems insisting on maximal parallelism means no numerically stable algorithm is known at all; in other cases there is a "smoother" tradeoff of parallelizability and stability. In these cases we propose the following paradigm for using unstable but fast algorithms safely:

1. Solve the problem (quickly).
2. Test for instability.
3. If the answer is unsatisfactory, recompute the answer using a slower but more stable algorithm.

This paradigm will be successful if the fast algorithm in the first step is only rarely unstable, and if the instability test in the second step is cheap. A number of problems can be solved this way, and are discussed in section 3.

In LAPACK as well as in many other packages we estimate condition numbers using fast approximation algorithms rather than attempting to compute them exactly. We do this because error bounds are approximations anyway, and high accuracy is generally not needed. Also, a user more interested in speed may not be willing to take much more time to compute an error bound on top of solving the original problem. However, the price of using estimators is that all of them invented so far have *counterexamples*, i.e. problems whose condition numbers are arbitrarily misestimated (usually underestimated). Fortunately, these counterexamples appear to be extremely rare. This leads us to the following conjecture:

> The complexity of computing a condition number with a certain guaranteed accuracy is at least as large as the complexity of the original problem.

In particular, this would imply that any fast estimator would necessarily have a counterexample. We will sketch a proof of this in a very simple (and nonrealistic) model of computation in section 4, and discuss an approach for analyzing how rare counterexamples are.

One of the goals of LAPACK was *portability of correctness*. Our need for machine specific versions of kernels like matrix-matrix multiplication (the Basic Linear Algebra Subroutines, or BLAS [33, 32, 31, 30]) in order to get high performance means that the potential

high speed of LAPACK is *not* portable without fast implementations of these kernels. But we strived very hard to make sure the computed answers are correct no matter how fast they are computed. As developers of putatively portable numerical software know, this is a hard problem because of the difference in floating point arithmetics provided on different machines, with different compilers, and with different basic mathematical function libraries. Fortunately, there is an opportunity to change this difficult situation, because of the widespread acceptance of IEEE floating point standard arithmetic [3]. There are numerous algorithms which would be much shorter and sometimes much faster if we were able to use certain features of IEEE arithmetic, especially the good rounding and the "sticky" exception flags. In section 5 we describe some of these algorithms.

2 High-Accuracy Linear Algebra Algorithms

We begin by describing the approach used to design and analyze the high accuracy algorithms already designed for LAPACK, and then discuss open problems.

We let H denote the problem for which we seek a solution for some problem; we denote the solution by $f(H)$. For example, $f(H)$ may denote the eigenvalues, eigenvectors, singular values, or singular vectors of the matrix H. If H denotes the pair (A, b), then $f(H)$ may denote the solution of the linear system $Ax = b$, perhaps in a least-squares sense if A is singular or not square. In general, $f(H)$ cannot be computed exactly and hence is approximated by an algorithm whose output we denote $\hat{f}(H)$. We also let ε denote the machine precision.

Analyzing the accuracy of an algorithm $\hat{f}$ for f consists of two parts. First, we use *perturbation theory*, where we bound the difference $f(H+\delta H)-f(H)$ in terms of δH. This part depends only on f and not the algorithm that approximates it. Second, we use *error analysis*, which attempts to show that the computed solution $\hat{f}(H)$ is close to $f(H+\delta H)$ for some bounded δH. Showing that $\hat{f}(H) = f(H+\delta H)$ for some bounded δH is called *backward error analysis*, but is by no means the only way to proceed.

There is a great deal of choice in the measures we choose to bound errors and measure distances. In conventional error analysis as developed by Wilkinson, we bound $\|f(H+\delta H)-f(H)\|$ in terms of $\|\delta H\|$, and show $\hat{f}(H) = f(H+\delta H)$ where $\|\delta H\| \le O(\varepsilon)\|H\|$. Here, $\|\cdot\|$ denotes a norm, like the one-norm or Frobenius norm. Typically one proves a formula of the form $\|f(H+\delta H)-f(H)\| \le \kappa(f,H)\cdot\|\delta H\| + O(\|\delta H\|^2)$, where $\kappa(H)$ is called the *condition number of H with respect to f*. In this formulation, it is easy to see that $\kappa(f,H)$ is simply the norm of the gradient of f at H: $\|\nabla f(H)\|$; other scalings are possible. Thus, combining the perturbation theory and error analysis, one can write

$$\|f(H+\delta H)-f(H)\| \le O(\varepsilon)\kappa(f,H)\cdot\|H\| + O(\varepsilon^2)$$

provided the algorithm is backward stable.

The drawback of this approach is that it does not respect the structure of the original data. In particular, if the original data is sparse or graded (large in some entries, small in others), bounding δH only by norm can give very pessimistic results. A trivial example is solving a diagonal system of equations. Each component of the solution is computed to full accuracy by a single divide operation, but the conventional condition number is the ratio of the largest to smallest diagonal entries and may be arbitrarily large.

Instead of bounding δH by its norm $\|\delta H\|$, one may instead use the measure $rel_H(\delta H) \equiv \max_{ij} |\delta H_{ij}|/|H_{ij}|$, the largest relative change in any entry (we use the notation rel_H to indicate the dependence on H). This measure respects sparsity, since δH_{ij} must be zero if H_{ij} is zero, and also grading, since every entry is perturbed by an amount small compared to its magnitude. For example, in the case of diagonal linear equation solving, one can easily see that a perturbation δH of size $rel_H(\delta H)$ in the matrix can only change the solution relatively by $rel_H(\delta H)$ in each component, and that the algorithm is backward stable with $rel_H(\delta H) \leq \varepsilon$. Thus, the new perturbation theory and error analysis with respect to $rel_H(\delta H)$ accurately predict that each component of the solution is computed to full relative accuracy.

We have successfully developed new perturbation theory, algorithms, and error analysis for the measure $rel_H(\delta H)$ for much of numerical linear algebra. We cannot always guarantee to solve problems as though we had a small $rel_H(\delta H)$, but the algorithms can in all cases monitor their accuracy and produce useful error bounds. The algorithms are usually small variations on conventional algorithms, perhaps with a slightly different stopping criterion, although the bidiagonal SVD algorithm has a quite new component. In all cases the algorithms run approximately as fast as their conventional counterparts, sometimes a little slower and sometimes a little faster. Since they are based on the conventional algorithms, all the techniques using the Level 3 BLAS apply to them.

This approach has been applied to linear equation solving [4], linear least-squares problems [5, 24, 58], the bidiagonal SVD [26, 16], the tridiagonal symmetric eigenproblem [61, 8], the dense symmetric positive definite eigenproblem [27], and the dense definite generalized eigenproblem [8, 27]. We have similar but slightly weaker results for the dense SVD and generalized SVD [27]. These algorithms either will be included directly in LAPACK or can be easily constructed by using LAPACK subroutines as "building blocks."

Now we discuss various open problems that remain to be solved, along with some indication of how hard we think they are.

High accuracy eigenvalues of Hessenberg matrices. The nonsymmetric eigenproblem has proven to be one of the more difficult problems to either parallelize successfully, or solve to high accuracy. Here we outline the building blocks that could be assembled into an algorithm that computes the eigenvalues of an upper Hessenberg matrix with tiny componentwise relative backward error. Hessenberg matrices are of interest because one can reduce a dense matrix to Hessenberg form quite effectively using matrix multiply and other BLAS [42, 36].

The first tool is a way to evaluate the determinant of an upper Hessenberg matrix H with tiny componentwise relative backward error. The method is due to Hyman, and is discussed in [90, p. 427], [89]. This method could be used in a Newton-based iterative refinement scheme to improve eigenvalues computed another way [37], or it could be used as the basis of a scheme itself [70]. To evaluate the accuracy of a computed eigenvalue or eigenvector pair, one can use an a posteriori estimate of the componentwise backward error; a simple expression for this error in given in [15], which is a simple generalization of a result of Oettli and Prager [74]. Finally, one needs a condition number. The simplest such expression is given as follows. Let λ be an eigenvalue of A with unit right eigenvector x (so $Ax = \lambda x$) and unit left eigenvector y^T (so $y^T A = \lambda y^T$). Suppose we perturb A by

δA, where $|\delta A_{ij}| \leq \eta |A_{ij}|$. Then to first order the perturbation $\delta\lambda$ in λ is given by [48]

$$\frac{|\delta\lambda|}{|\lambda|} \leq \frac{|y^T \delta A x|}{|\lambda y^T x|} = \frac{|y^T \delta A x|}{|y^T A x|} \leq \eta \cdot \frac{|y^T| \cdot |A| \cdot |x|}{|y^T A x|} \quad ,$$

a quantity we easily to see be at least η, and exceeding η to the extent we have cancellation in the evaluation of $y^T A x$. Thus this provides a condition number with which we can compute relative error bounds for computed eigenvalues.

Stability of Parallel Prefix Operation. Suppose $x_1, \ldots, x_n$ are data items, and $\oplus$ is an associative operator acting on them. We wish to compute $y_1, \ldots, y_n$ where $y_i = x_1 \oplus \cdots \oplus x_i$. It turns out all the y_i can be computed on $O(\log_2 n)$ time using a single tree of processors; this operation is called *parallel prefix* [68, 22, 11, 13]. A large number of important computations can be reformulated as parallel prefix operations, and in fact Kung has shown that all rational scalar recurrences ($x_{i+1} = f_i(x_i)$ where f_i is a rational function of the scalar x_i) which can be parallelized at all using rational operations can be parallelized using parallel prefix where the associative operation is 2-by-2 matrix multiplication [67]. For example, the eigenvalues of a symmetric tridiagonal matrix T with diagonal entries $a_1, \ldots, a_n$ and offdiagonal entries $b_1, \ldots, b_{n-1}$ can be found using the Sturm sequence

$$d_i = (a_i - \sigma) d_{i-1} - b_{i-1}^2 d_{i-2}$$

where d_i is the determinant of the leading i-by-i principal submatrix of $T - \sigma I$. By Sylvester's theorem the number of sign changes in the sequence of d_i's is the number of eigenvalues of T less than σ. This can be used to count the number of eigenvalues in any interval $[\sigma_1, \sigma_2]$ and so find all the eigenvalues of T by bisection [51]. This scheme is very stable numerically if evaluated serially in $O(n)$ time. It can be evaluated in $O(\log_2 n)$ using parallel prefix by rewriting it as

$$\begin{bmatrix} d_i \\ d_{i-1} \end{bmatrix} = \begin{bmatrix} a_i - \sigma & -b_{i-1}^2 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} d_{i-1} \\ d_{i-2} \end{bmatrix} \equiv M_i \begin{bmatrix} d_{i-1} \\ d_{i-2} \end{bmatrix} = M_i \cdot M_{i-1} \cdots M_1 \cdot \begin{bmatrix} d_0 \\ d_{-1} \end{bmatrix}$$

This technique, or ones like it, have been suggested in [64, 85], where good numerical results have been attained. But so far no one has succeeded in proving it is stable, and it appears difficult to do so. Also, the paradigm we proposed earlier for using possibly unstable algorithms, checking quickly for instability and recomputing if necessary, is hard to apply because we know of no faster way to confirm the accuracy of an eigenvalue than running this parallel prefix operation. So studying the numerical stability of parallel prefix is an important open problem.

Accuracy of Jacobi's Method. In [27] it was shown that Jacobi's method (with a suitable stopping condition) for finding the eigenvalues of a symmetric positive definite matrix could be much more accurate than other methods based on tridiagonalization followed by solving the tridiagonal eigenproblem; similar results were obtained for the SVD. The reason for this is as follows: Let H be the symmetric positive definite matrix whose eigenvalues we desire. Write $H = DAD$, where $D = \text{diag}\,(H_{11}^{1/2}, \ldots, H_{nn}^{1/2})$, and $A_{ii} = 1$. One can show that this diagonal scaling of H results in A having a condition number $\kappa(A) \equiv \|A\|_2 \cdot \|A^{-1}\|_2$ never much larger than $\kappa(H)$ and potentially much smaller, especially if the H_{ii} vary greatly in

magnitude. One can show that a *normwise* perturbation in H of size $\eta \ll 1$ can cause relative changes in the eigenvalues of at most about $\eta\kappa(H)$, but that a *componentwise relative* perturbation in H of size η can cause relative changes in the eigenvalues of at most $\eta\kappa(A)$, which can be much smaller. Furthermore, one step of Jacobi can also only change the eigenvalues by a relative amount of size $\eta\kappa(A)$, so that the errors introduced by *one step* of Jacobi are no worse than tiny componentwise relative error in the original data.

Of course, Jacobi does not converge in one step. Let $H = H_0 = D_0 A_0 D_0$ be the initial matrix and its factorization as above. Let H_i be the matrix after the i-th Jacobi rotation, and let $H_i = D_i A_i D_i$ be its analogous factorization. Eventually H_i approaches a diagonal matrix Λ of eigenvalues, D_i approaches $\Lambda^{1/2}$ and A_i approaches the identity matrix I. The relative error in the eigenvalues cause by the i-th Jacobi step is bounded by $O(\varepsilon)\kappa(A_i)$, so the overall error of the algorithm is bounded by $O(\varepsilon)\max_i \kappa(A_i)$. Since the minimum possible error bound, due to small relative changes in the initial data, is $O(\varepsilon)\kappa(A_0)$, the ultimate accuracy of Jacobi depends on how much larger $\max_i \kappa(A_i)$ can be than $\kappa(A_0)$. Note that since A_i eventually approachs I, $\kappa(A_i)$ approaches 1, and so it is the transient rather than the asymptotic behavior of $\kappa(A_i)$ that is of interest.

In many thousands of numerical experiments on random matrices, the ratio $\max_i \kappa(A_i)/\kappa(A_0)$ never exceeded 1.82. This means the error bound attained by Jacobi is nearly as good as the best possible one. Further work by Mascarenhas [73] found a family of examples where $\max_i \kappa(A_i)/\kappa(A_0)$ can be as large as $n/2$, but this is not bad since there are factors of n or more in the $O(\varepsilon)$ factor anyway. Also in [21] we showed that tridiagonal QR iteration can fail to compute eigenvalues to high relative accuracy because there are cases where $\max_i \kappa(A_i)/\kappa(A_0)$ is as large as $1/\varepsilon$. Thus, the factor $\max_i \kappa(A_i)/\kappa(A_0)$ plays a central role in predicting the accuracy with which we can compute eigenvalues and understanding when it is small is of interest.

General Structured Backward Error. The real goal of a user of a numerical algorithm may not so much be tiny componentwise relative backward error in the solution of the numerical model, but rather tiny backward error in the original physical problem. For example, one might want to solve a differential equation with tiny backward error, and this may or may not be implied by solving the corresponding discretized problem with tiny backward error. Depending on how the parameters of the physical problem appear in the discrete model, it may be quite hard to even compute the backward error. If there are more output parameters than input parameters, it will generally be impossible to achieve tiny backward error for dimensional reasons. Even the simple problem of solving $Ax = b$, $A = A^T$, with tiny componentwise relative *symmetric* backward error turns into a high dimensional sparse underdetermined least squares (or least l_∞) problem, with no apparently simple solution [53, 9].

3 Parallel Algorithms

In this section we list problems where we still need good parallel algorithms, even ones stable in the conventional normwise sense. There is of course a tremendous amount of activity in this area, so we will limit ourselves to problems that have arisen in the course of work on LAPACK. In particular, we will limit ourselves to direct algorithms for dense problems.

The algorithms currently under investigation illustrate the paradigm of the introduction: since they can be unstable, their use requires the ability to quickly determine their stability, perhaps a posteriori, and to recompute the answer stably if required.

The nonsymmetric tridiagonal eigenvalue problem. Here the approach is to reduce a dense nonsymmetric matrix to tridiagonal form via nonorthogonal transformations, and then solve the resulting nonsymmetric tridiagonal eigenproblem [34, 45, 46, 47]. The main difficulty is that the similarity which reduces a matrix to tridiagonal form can be arbitrarily ill-conditioned, and in fact one need not exist at all. The advantage is that it is cheaper to find the eigenvalues of a nonsymmetric tridiagonal matrix than a Hessenberg one.

The Hessenberg eigenvalue problem. As stated above, there are good block algorithms for reducing a dense nonsymmetric matrix to upper Hessenberg form, and several algorithms begin with this form. The standard serial algorithm for this problem is the Francis QR algorithm [51], which produces a sequence H_i of orthogonally similar Hessenberg matrices which converge to Schur form. To compute H_{i+1} from H_i, one performs row and column operations starting from one end of the matrix and working towards the other. This process is called "chasing the bulge" since at any intermediate point there is a bulge, or triangle of nonzero entries lying below the subdiagonal of H_i and spoiling its Hessenberg structure. By increasing the size of this bulge, one can perform matrix-vector (Level 2 BLAS) operations instead of vector-vector operations (Level 1 BLAS) but the speed up available is modest [6, 39].

There are two other techniques to reduce the Hessenberg problem to a series of simpler problems: tearing and homotopy. Tearing [37] involves setting a subdiagonal entry of H near the middle to zero, thus forming two independent upper Hessenberg eigenproblems which can be solved in parallel. Given the solutions to these problems, they must be merged to yield the solution of the entire matrix; clearly this approach may be applied recursively to the smaller subproblems encountered. This approach has been applied with great success to the symmetric tridiagonal eigenproblem, where the merging process yields a scalar *secular equation* to solve, a simple rational function whose roots are the eigenvalues, and for which monotonically and globally quadratically convergent Newton based iterative methods exist [14, 38, 82]. The method even provides disjoint intervals in which the function whose zero we desire is monotonic and guaranteed to have a single unique solution.

The Hessenberg problem is significantly harder. First, the eigenvalues are complex, and there is no guaranteed convergent iteration or even a simple way to localize the desired roots. The eigenvectors as well as the eigenvalues must be computed, and all these may be very ill-conditioned, and sometimes not even exist. Even if the initial problem has well-conditioned eigenvalues and eigenvectors, smaller intermediate problems may be very badly conditioned. If two or more different Newton iterations seem to converge to the same root, it is hard to tell if the root is really multiple or if another root is not being found [59].

The homotopy method can be though of as variant of the above scheme, where one (or more) subdiagonals are set to zero, the resulting simpler subproblems solved (perhaps recursively), and then the solutions merged by gradually increasing the zero subdiagonals to their original values and following the curves of eigenvalues (and possibly eigenvectors) from their original values as eigenvalues of subproblems to eigenvalues of the original problem. This curve following can be done in many ways, such as predictor corrector where one

predicts using Euler's method and corrects using Newton. One would usually try a large stepsize first, such as going all the way to the final solution in one step (in which case this method is very similar to the previous one), and only taking smaller steps if necessary. This method is also hard to stabilize, since if the curves become very close very tiny step sizes are needed to distinguish them, or else stability can be lost [70].

The dense nonsymmetric eigenproblem – divide and conquer. Instead of initially reducing the dense matrix A to a condensed form like Hessenberg or tridiagonal, one can instead work directly on A [71]. These approaches try to divide and conquer the problem by computing an orthogonal matrix $U = [U_1, U_2]$ where the columns of U_1 (approximately) span an invariant subspace of A, so that

$$U^T A U = \begin{bmatrix} U_1^T A U_1 & U_1^T A U_2 \\ U_2^T A U_1 & U_2^T A U_2 \end{bmatrix} \equiv \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix}$$

is nearly block upper triangular, i.e. A'_{21} is nearly zero. If A'_{21} is small enough, one can just find the eigenvalues of A'_{11} and A'_{22}, perhaps recursively. So how might one find such an orthogonal U? The idea is to find a simple rational map f which maps one subset $\mathcal{S}$ of the complex plane to (or near) one point s, and (the interior of) its complement $\mathcal{S}'$ to another point s'. Then $f(A)$ will (approximately) have two eigenvalues s and s', and so $f(A) - s'I$ will have eigenvalues $s - s'$ and 0. Then, provided there are no 2-by-2 or larger Jordan blocks associated with eigenvalue 0, the columns of $f(A) - s'I$ will span the invariant subspace associated with all eigenvalues inside region $\mathcal{S}$.

One choice of f is the *sign function*: $\mathrm{sign}(z) = 1$ if $\Re z > 0$, $\mathrm{sign}(z) = 0$ if $\Re z = 0$, and $\mathrm{sign}(z) = -1$ if $\Re z < 0$. This may be extended to a function of matrices in the usual way, provided there are no pure imaginary eigenvalues. Thus if

$$A = [X_1, X_2] \begin{bmatrix} J_{11} & 0 \\ 0 & J_{22} \end{bmatrix} [Y_1, Y_2]^T$$

is the Jordan canonical form of A, where $[Y_1, Y_2]^T = [X_1, X_2]^{-1}$, the eigenvalues of J_{11} are in the open left half plane and the eigenvalues of J_{22} in the open right half plane, then

$$\mathrm{sign}(A) = [X_1, X_2] \begin{bmatrix} -I & 0 \\ 0 & I \end{bmatrix} [Y_1, Y_2]^T \ .$$

Thus $\mathrm{sign}(A) - I = -2X_1 Y_1^T$, and its column space spans the invariant subspace of A associated with eigenvalues in the left half plane.

It turns out that there is a very simple globally convergent and asymptotically quadratically convergent iteration for computing $\mathrm{sign}(A)$: $A_{i+1} = .5(A_i + A_i^{-1})$. Once A_i is close enough to its limit the following scheme is also quadratically convergent, and avoids inversion: $A_{i+1} = .5A_i(3I - A_i^2)$. Other higher order schemes are known too, but they are more expensive to evaluate, and round off tends to obscure the small eigenvalues of powers of a matrix, so it is not clear that these schemes help.

There are certainly open problems associated with this scheme. In order to divide the spectrum nearly in half each time, one might try to choose a shift σ so close to half the spectrum of $A - \sigma I$ is in the left half plane and half in the right. If most of the eigenvalues

have nearly the same real part (such as a skew symmetric matrix, all of whose eigenvalues are pure imaginary), then no splitting is possible, and one might consider squaring the matrix to rotate the spectrum. If the eigenvalues lie very close together on one dimensional curves, as is the case in many applications, then no shifting or squaring scheme will leave a gap around the imaginary axis, which is needed for fast convergence to the sign function. Still, the method has attractions, such as being able to use basic building blocks such as matrix multiplications, inversion, QR decomposition (to compute U_1 from the columns of $\text{sign}(A) - I$), and so on. In addition, the fact that it always acts on the original data by multiplication by orthogonal matrices means it is numerically stable, provided we iterate until A'_{21} is sufficiently small.

The dense nonsymmetric eigenproblem – Jacobi. Jacobi's method has been generalized to apply to dense nonsymmetric matrices [40, 41, 75, 78, 79, 83, 87, 88]. The parallelism arises in the ability to apply rotations to disjoint pairs of rows or columns in parallel. Some authors [41, 83] consider only orthogonal (or unitary) transformations, and try to converge to the Schur form. Others [75, 78, 79] use nonunitary transformations as well, and try to converge to diagonal form, provided the matrix is diagonalizable. The unitary methods guarantee numerical stability, but appear to only be asymptotically linearly convergent. The nonunitary methods can be made to be asymptotically quadratically convergent, but cannot guarantee backward stability. Still, these methods tends to move the original matrix closer to a normal matrix, which has well-conditioned eigenvalues, and so in practice the errors do not seem much worse than the condition number warrants. Convergence tends to slow down the farther from normal the matrix is. Most questions about this class of methods are open: global convergence, retaining real arithmetic if the original matrix is real [78, 87, 88], and avoiding instability and simultaneously slowdown of convergence for highly nonnormal matrices.

The generalized nonsymmetric eigenproblem. All of the above algorithms and challenges apply even more to the generalized regular eigenvalue problem $A - \lambda B$. Regularity means $A - \lambda B$ is square and has a determinant which is not identically zero. Such $A - \lambda B$ have n finite or infinite eigenvalues; for the more general case see [43, 86, 25, 84]. The standard serial algorithm [81] first reduces A to upper Hessenberg form and B to upper triangular form; we do not even have a block algorithm based on matrix-vector or matrix-matrix operations for performing this reduction. The extension of the other techniques mentioned above has not yet been attempted.

4 The Complexity of Condition Estimation

Fast estimators of condition numbers are ubiquitous in numerical linear algebra, because they provide inexpensive error bounds, and are quite reliable [54, 1]. Still, counterexamples are known for all existing estimators, i.e. matrices for which the estimators underestimate the true condition numbers by arbitrary amounts. Thus, research continues on making estimators yet more reliable while retaining their low complexity. Based on this experience, we make the following

Conjecture: The complexity of estimating a condition number with a guaranteed error bound is as large as solving the original problem.

We make this rather vague conjecture more precise in the case of linear equation solving: computing an estimate of $\|A^{-1}\|$ in any norm with any guaranteed accuracy at all is as difficult as computing A^{-1} itself. In particular, to compute $\|A^{-1}\|$ with any accuracy, we clearly need to decide whether A is singular, so condition estimation is at least as hard as deciding singularity. We outline a proof that deciding whether $\det(A) = 0$ is as hard as computing A^{-1} in the following very simple (and nonrealistic) model of computation: We suppose the entries of A are complex numbers, run a straight-line program which can perform any of the four basic operations $+$, $-$, $\times$ and $\div$ but which is not allowed to divide by zero for any input A, and compare the resulting rational function $f(A)$ of the entries of A to zero. The function $f(A)$ must clearly be an integer power k of $\det(A)$, since the determinant is an irreducible polynomial, and the presence of any other polynomial factor in $f(A)$ would for some A either lead to an incorrect decision that A is singular (if it appears in the numerator of A), or to division by zero (if it appears in the denominator). Now note that by Cramer's rule, each entry of A^{-1} may be written $(\partial f(A)/\partial A_{ij})/(k \cdot f(A))$. By a result in [10], there exists a straightline code that computes all the $\partial f(A)/\partial A_{ij}$ in three times the number of nontrivial multiplies and divides needed to compute $f(A)$. Then all the $(\partial f(A)/\partial A_{ij})/(j \cdot f(A))$ can be computed in $n^2 + 1$ more steps, which less than doubles the operation count so far by a fan-in argument.

More practically, we would like to assess the reliability of a particular estimation scheme, or to be able to compare two schemes. For example, in [65] the authors compare two estimators for $\|A\|_2$ for an n-by-n symmetric positive definite matrix A, where one is permitted only to multiply A by an arbitrary vector. This could in principal be used to estimate the smallest singular value of a general matrix G since $\sigma_{\min}(G) = \|(GG^T)^{-1}\|_2^{-1/2}$, and multiplying by $(GG^T)^{-1}$ can be done cheaply given the LU factorization of G. The authors compare k steps of the power method and k steps of Lanczos applied to this problem with a random starting vector x_0. They show that the probability that the relative error in the estimate of $\|A\|_2$ exceeds e is at most $\sqrt{n}(1-e)^k$ for the power method, and at most $\sqrt{n}\exp(-\sqrt{e}(2k-1))$ for Lanczos. Thus, for small e, Lanczos has a much lower probability of its relative inaccuracy exceeding e than the power method; this is another way to express the fact that Lanczos extracts the maximum information from the Krylov basis $[x_0, Ax_0, A^2x_0, \ldots, A^kx_0]$ whereas the power method does not. Another probabilistic analysis of the power method appears in [28].

A different probabilistic approach for comparing methods is as follows. It is motivated by the approach in [18, 19], where ill-conditioning is associated with nearness to a particular algebraic variety, and then the chance that a random problem lies close to that variety is estimated using just the degree and codimension of the variety. In the case of condition estimation one would try to show that the estimator worked well unless the matrix lay close to a particular variety (or perhaps semialgebraic set), and then estimate the chance that a random problem lay close to that variety. This approach would not distinguish between the power method and Lanczos, since they both use the same basic information about the matrix (its projection on a Krylov subspace) but might be able to distinguish among the plethora of other estimators schemes which have been proposed.

5 Exploiting Good Floating Point Arithmetic

A great deal of effort went into trying to make LAPACK portably correct despite the variations in floating point arithmetics, compiler optimizations, and ability to handle exceptions. Our goal was to write "mail order software" that would work correctly even if passed around in source form from machine to machine, since that is the model of software development supported by systems like netlib. We did not attempt to get equally portable high performance, since the BLAS are machine dependent, as well as the optimal block sizes used by each subroutine.

In particular, we had to make worst case assumptions about the floating point environments:

1. Rounding is sloppy, done without guard digits, so that for any operation $\oplus \in \{+, -, \times, \div\}$, one can say only that the rounded value of $a \oplus b$ is

$$fl(a \oplus b) = (a(1 + \delta_1)) \oplus (b(1 + \delta_2))$$

   where $|\delta_1|$ and $|\delta_2|$ are both bounded by some tiny ε. ε need not be as small as the relative difference between adjacent floating point numbers.

2. All exceptions except underflow are possibly fatal and to be avoided as much as is reasonable, in particular when the final result consists of a representable floating point number.

3. It is possible for complex division and the Level 1 BLAS routine xNRM2 [69] for computing the Euclidean length of a vector to malfunction when some of the data exceeds the square root of overflow, or is nonzero but all less than the square root of underflow.

4. No mixed precision is permitted, since a single precision code using some double precision can not be simply translated to a double precision code since quadruple precision is not generally available.

We also, out of exasperation, made several assumptions which are actually violated by existing machines, because they only made a few test cases fail and because taking them into account would have significantly complicated or slowed down the software:

1. It is safe to compute x/y if $0 < x \leq y$ without fear of exception. However, this operation can overflow on machines like Crays which actually compute $x * (1/y)$.

2. The underflow threshold is significantly smaller than ε^3. This is not true in Vax D-format, and causes failures we still do not completely understand in some badly scaled test cases.

However, the price paid for this portable correctness was high, with penalties in speed, functionality, accuracy and software productivity. In contrast, if we had been able to assume a uniform floating point environment with IEEE floating point arithmetic [3], these problems would have been avoided. Not all features of IEEE arithmetic are needed to avoid each problem; in some cases any reasonably carefully rounded arithmetic would have done,

whereas in others certain details of IEEE exception handling are important. In particular, a large part of the potential benefit is attainable with an efficiently implementable subset of the possible exception handling mechanism suggested by the IEEE standard [23]. This is important because full IEEE compliance in supplying precise interrupts is likely to be inefficient on the heavily pipelined architectures common even in microprocessors.

Here we will discuss the penalties paid for portability, and how IEEE conformance alleviates them. Since virtually all machines being built or on the drawing boards conform at least partially to IEEE arithmetic, we believe the time has come to write software specialized to IEEE arithmetic (or at least some features of IEEE). Otherwise, the great investment in hardware made by manufacturers will not pay off in faster, more accurate, more reliable and more speedily written software, which was the entire motivation of the standard.

Computing machine constants – loss of software productivity. Subroutine SLAMCH in LAPACK computes basic machine constants like the machine precision ε, underflow threshold, overflow threshold, the base, rounding style, and so on. It is 339 lines of Fortran long (not counting comments), and was quite difficult to write. It can be thought of as a simplified version of programs like Paranoia [60] (over 2300 lines of Fortran without comments) which attempt to characterize the details of machine arithmetic (as seen through a high level language). Most algorithms need only reasonably accurate values of the machine precision, over and underflow threshold in order to work correctly, and for these SLAMCH is adequate. However, for the more subtle algorithms discussed below it is difficult or impossible to reliably discover at run time whether the arithmetic, compiler, and mathematical libraries have the necessary properties for the algorithms to work. For example, simulating double precision requires the basic rounding to be accurate enough [77, 17, 72] and although this can be tested at runtime it is very time consuming and not foolproof [60]. Determining how underflow is handled (or even finding the exact overflow threshold) requires causing an overflow, and this may be fatal. Many more examples can be cited.

There are two approach to this problem. In the short run we will simply be assuming IEEE arithmetic, in which all these features are well defined (even if the software interface to exception handling is not). In the long run people are, for better or worse, likely to continue inventing new styles of floating point, as well as languages and compilers providing new interfaces, expression evaluation mechanisms, parameter passing mechanisms, and other features impacting the floating point environment as perceived by the programmer [49, 62, 63, 66, 76]. These developments are guaranteed to repeatedly make any program like SLAMCH obsolete. It would be nice to have a standard set of environmental enquiries describing all possibly relevant details of the arithmetic, which could be supplied by the compiler implementor. But it is difficult to imagine a terse but complete set of such enquiries at the moment, and tentative steps in this direction have not succeeded [62, 76].

Condition estimation and eigenvector computation – loss of speed. Both these computations involve solving triangular systems of equation. To estimate $\|A^{-1}\|$, one can use an LU factorization of A to repeatedly solve either $Ax = b$ or $A^T x = b$ for certain cleverly chosen b [52, 54, 56, 55, 57]. To compute eigenvectors one can either reduce to

Schur form

$$T = \begin{bmatrix} T_{11} & t_{12} & T_{13} \\ 0 & \lambda & t_{23} \\ 0 & 0 & T_{33} \end{bmatrix}$$

and solve a triangular system $(T_{11} - \lambda I)x = -t_{12}$ to find the right eigenvector $[x^T, 1]^T$ of λ explicitly, or else reduce just to Hessenberg form H and do inverse iteration $(H - \lambda I)x_{i+1} = x_i$ using an LU factorization of $H - \lambda I$ [51].

Thus, it appears that one could just use the Level 2 BLAS routine [33, 32] for solving triangular systems in all these cases. Unfortunately, we can not, because in all cases we anticipate solving ill-conditioned systems which could lead to overflow. In the case of condition estimation, we want a condition estimate as a warning of potential problems in solving $Ax = b$, and in particular we would want a warning if overflow is possible, since overflow is generally fatal and to be avoided. If we are computing eigenvectors by solving $(T_{11} - \lambda I)x = -t_{12}$, then if λ is (nearly) an eigenvalue of T_{11} too, the system will be very ill-conditioned and overflow will be possible. When solving $(H - \lambda I)x_{i+1} = x_i$ the more accurate λ is, the more singular $H - \lambda I$ will be and the more likely overflow will be. In both these cases overflow is not a warning signal to the user, but rather an internal event of no interest to the user.

To deal with potential overflow, we had to write new versions of all the triangular solvers in LAPACK which scaled in the innermost loop to avoid overflow. To see that we can not simply scale T and b to solve $Tx = b$ consider an n-by-n upper bidiagonal T with ones on the superdiagonal and ε's on the diagonal; then T^{-1} has an entry of size ε^{-n} which can be much larger than the overflow threshold. To see how complicated this may be, the LAPACK subroutine SLATRS (which deals both with $Tx = b$ and $T^Tx = b$) is 300 lines of Fortran (not counting comments).

If we were only interested in condition estimation, as in subroutine SGECON, an overflow would signal extreme ill-conditioning and in fact let us stop immediately, returning RCOND (estimated reciprocal condition number) equal to zero. With the sticky overflow flag of IEEE arithmetic, this would be possible by simply calling the Level 2 BLAS triangular solver and testing the overflow flag on return. This lets the code go at its top possible Level 2 BLAS speed, and still be robust.

For eigenvector computations, we currently see no way to avoid sophisticated scaling in some cases, but for the majority of cases where overflow does not occur, we could again run at the top speed of Level 2 BLAS and not pay the insurance premium of scaling in the inner loop unless required. This speed-up can be significant on some machines.

Divide and conquer algorithm for the symmetric tridiagonal eigenproblem – loss of functionality. The algorithm proposed in [14] and further developed in [38] is a fast parallel method for the symmetric tridiagonal eigenproblem, and can be much faster than the older QR method even on serial machines. However, it is not stable unless great care is taken in solving the secular equation, a rational equation whose roots are the eigenvalues at each step merging the solutions of two subproblems. In fact, in [82] it is shown that double precision solution of the secular equation (double in the input precision, whatever that is) is in fact *necessary* in order to have a stable algorithm. The amount of double precision needed is small, and so even if double is simulated somewhat inefficiently it will not affect

the overall speed. Certainly a factor of 10 slowdown over single would not be too bad.

There are three ways to get access to double precision. One is simply to declare some double precision variables, but this violates our ban on mixed arithmetic. One could also simulate double using integer arithmetic, but the known methods for doing this portably [12, 7] are quite slow indeed for double precision, although they are useful for very high precision. This leaves one with simulating double precision using single precision operations, for which many efficient techniques are known [77, 17, 72]. But these techniques require sufficiently accurate floating point arithmetic, in particular guard digits.

If we knew arithmetic were correctly rounded, and if we knew how many digits of accuracy there were, then we could simulate double precision using single perhaps only 5 or 6 times more slowly than just using single alone. We will use this technique for the divide and conquer algorithm for the symmetric tridiagonal eigenproblem.

Parallel bisection for the symmetric tridiagonal eigenproblem – loss of speed. Let T be a symmetric tridiagonal matrix with diagonal entries $a_1, \ldots, a_n$ and off diagonal entries $b_1, \ldots, b_{n-1}$. As stated in a previous section the Sturm sequence $d_i = (a_i - \sigma)d_{i-1} - b_{i-1}^2 d_{i-2}$ can be used to count the number of eigenvalues of T less than σ. We may evaluate this recurrence in $O(\log_2 n)$ time using parallel prefix with the operation of 2-by-2 matrix multiplication with matrices of the form $M_i = \begin{bmatrix} a_i - \sigma & -b_{i-1}^2 \\ 1 & 0 \end{bmatrix}$. The difficulty is that the entries of products of M_i matrices grow or shrink essentially as fast as determinants of submatrices of T_i, and so are very prone to over/underflow. One could scale within the inner loop, but this would be slow for the same reason condition estimation is currently slow. Again, an overflow and an underflow flag would be quite helpful. In fact, the *wrapped exponent* feature of IEEE arithmetic would be particularly helpful, because it returns the true value of an underflowed or overflowed quantity with the exponent biased up or down by a known amount to keep the returned result in range [3]. This can be used to good effect to speed up the calculation [22].

2 by 2 and other subproblems – loss of accuracy and software productivity. In LAPACK much effort was expended on building highly reliable routines for the small (2 by 2 and somewhat larger) linear systems and eigenproblems that must be solved within larger solvers. These small routines need to be highly reliable since they form the kernels of the larger problem. It may seem surprising that such small problems were so difficult to solve well, but this experience is corroborated by the experience of other developers of linear algebra software. If, on the other hand, we had been able to assume (simulated) double precision when necessary, vast simplifications would have been possible. For example, subroutine SLAS2 computes the eigenvalues of a 2-by-2 triangular matrix, and gets them accurately no matter their values on virtually any machine we know of. It is 33 (nontrivial) lines of Fortran long. In contrast, a code using simulated double could in principal be 3 (nontrivial) lines long, depending on how much syntactic sugar was available to access the simulated double. The same comments apply to many other routines as well. In the first public release of LAPACK, we did not include routines for the generalized nonsymmetric eigenproblem $A - \lambda B$, partly because the corresponding 2 by 2 routines were so difficult to write. For the next LAPACK release, we plan to write these assuming simulated double is available.

Iterative refinement – loss of accuracy. The expert drivers for solving linear systems in LAPACK do single precision iterative refinement to improve the solution [4, 80]. This refinement will under certain technical assumptions guarantee a tiny componentwise relative backward error. In contrast, the earlier conventional wisdom had held that computing residuals in double precision was needed to justify the procedure, in which case one could guarantee a tiny forward error provided the problem was not truly badly conditioned. So double precision residual accumulation has definite advantages, but is not available to us if we eschew mixed precision. Of course, being able to use simulated double makes it available again, as well as an array of other iterative refinement schemes based on double precision iterative refinement. Since the cost of the double precision part of the calculation (for dense problems) is $O(n^2)$ in contrast to the remaining $O(n^3)$ part, the marginal cost of this refinement is small, and so it is well worth doing.

References

[1] 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 1.0.* SIAM, Philadelphia, 1992.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK: A portable linear algebra library for high-performance computers. Computer Science Dept. Technical Report CS-90-105, University of Tennessee, Knoxville, 1990. (LAPACK Working Note #20).

[3] ANSI/IEEE, New York. *IEEE Standard for Binary Floating Point Arithmetic*, Std 754-1985 edition, 1985.

[4] M. Arioli, J. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.*, 10(2):165–190, April 1989.

[5] M. Arioli, I. S. Duff, and P. P. M. de Rijk. On the augmented system approach to sparse least-squares problems. *Num. Math.*, 55:667–684, 1989.

[6] Z. Bai and J. Demmel. On a block implementation of Hessenberg multishift QR iteration. *International Journal of High Speed Computing*, 1(1):97–112, 1989. (also LAPACK Working Note #8).

[7] D. H. Bailey. MPFUN: A portable high performance multiprecision package. Technical Report RNR-90-022, NASA Ames Research Center, 1990. submitted for publication.

[8] J. Barlow and J. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Num. Anal.*, 27(3):762–791, June 1990.

[9] S.G. Bartels and D.J. Higham. The structured sensitivity of vandermonde-like systems. Technical report, Numerical Analysis Report NA/134, University of Dundee, 1991. to appear in Numer. Math.

[10] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1982.

[11] G. Belloch. Prefix sums and their applications. School of Computer Science Technical Report CMU-CS-90-190, Carnegie Mellon University, November 1990.

[12] R. P. Brent. A Fortran multiple precision arithmetic package. *ACM Trans. Math. Soft.*, 4:57–70, 1978.

[13] S. Chatterjee, G. Belloch, and M. Zagha. Scan primitives for vector computers. In *Proceedings of Supercomputing '90*, New York, NY, 1990.

[14] J.J.M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36:177–195, 1981.

[15] A. Deif. A relative backward perturbation theorem for the eigenvalue problem. *Num. Math.*, 56:625–626, 1989.

[16] P. Deift, J. Demmel, L.-C. Li, and C. Tomei. The bidiagonal singular values decomposition and Hamiltonian mechanics. *SIAM J. Num. Anal.*, 28(5):1463–1516, October 1991. (LAPACK Working Note #11).

[17] T. Dekker. A floating point technique for extending the available precision. *Num. Math.*, 18:224–242, 1971.

[18] J. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Num. Math.*, 51(3):251–289, July 1987.

[19] J. Demmel. The probability that a numerical analysis problem is difficult. *Math. Comput.*, 50(182):449–480, April 1988.

[20] J. Demmel. CS 267 Course Notes: Applications of Parallel Processing. Computer Science Division, University of California, 1991.

[21] J. Demmel. The inherent inaccuracy of implicit tridiagonal QR. Technical report, IMA, University of Minnesota, 1992.

[22] J. Demmel. Specifications for robust parallel prefix operations. Technical report, Thinking Machine Corp., 1992.

[23] J. Demmel, J. Dongarra, and W. Kahan. On designing portable high performance numerical libraries. In *1991 Dundee Numerical Analysis Conference Proceedings*, Dundee, Scotland, June 1991.

[24] J. Demmel and N. J. Higham. Improved error bounds for underdetermined system solvers. Computer Science Dept. Technical Report CS-90-113, University of Tennessee, Knoxville, 1990. (LAPACK Working Note #23; to appear in SIAM J. Mat. Anal. Appl.).

[25] J. Demmel and B. Kågström. Stable eigendecompositions of matrix pencils $A - \lambda B$. Institute of Information Processing Report UMINF-118.84, University of Umeå, Umeå, Sweden, 1984.

[26] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, September 1990.

[27] J. Demmel and K. Veselić. Jacobi's method is more accurate than QR. Computer Science Dept. Technical Report 468, Courant Institute, New York, NY, October 1989. (also LAPACK Working Note #15), to appear in SIAM J. Mat. Anal. Appl.

[28] J. D. Dixon. Estimating extremal eigenvalues and condition numbers of matrices. *SIAM J. Num. Anal.*, 20:812–814, 1983.

[29] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart. *LINPACK User's Guide.* SIAM, Philadelphia, PA, 1979.

[30] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16(1):18–28, March 1990.

[31] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[32] J. Dongarra, J. Du Croz, S. Hammarling, and Richard J. Hanson. Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subroutines. *ACM Trans. Math. Soft.*, 14(1):18–32, March 1988.

[33] J. Dongarra, J. Du Croz, S. Hammarling, and Richard J. Hanson. An extended set of fortran basic linear algebra subroutines. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[34] J. Dongarra, G. A. Geist, and C. Romine. Computing the eigenvalues and eigenvectors of a general matrix by reduction to tridiagonal form. Technical Report ONRL/TM-11669, Oak Ridge National Laboratory, 1990.

[35] J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5):403–407, July 1987.

[36] J. Dongarra, S. Hammarling, and D. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. *JCAM*, 27:215–227, 1989. (LAPACK Working Note #2).

[37] J. Dongarra and M. Sidani. A parallel algorithm for the non-symmetric eigenvalue problem. Computer Science Dept. Technical Report CS-91-137, University of Tennessee, Knoxville, TN, 1991.

[38] J. Dongarra and D. Sorensen. A fully parallel algorithm for the symmetric eigenproblem. *SIAM J. Sci. Stat. Comput.*, 8(2):139–154, March 1987.

[39] A. Dubrulle. The multishift QR algorithm: is it worth the trouble? Palo Alto Scientific Center Report G320-3558x, IBM Corp., 1530 Page Mill Road, Palo Alto, CA 94304, 1991.

[40] P. Eberlein. A Jacobi method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix. *J. SIAM*, 10:74–88, 1962.

[41] P. Eberlein. On the Schur decomposition of a matrix for parallel computation. *IEEE Trans. Comput.*, 36:167–174, 1987.

[42] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh. Parallel algorithms for dense linear algebra computations. *SIAM Review*, 32:54–135, 1990.

[43] F. Gantmacher. *The Theory of Matrices, vol. II (transl.)*. Chelsea, New York, 1959.

[44] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[45] G. A. Geist. Reduction of a general matrix to tridiagonal form. Technical Report ONRL/TM-10991, Oak Ridge National Laboratory, 1989.

[46] G. A. Geist and G. J. Davis. Finding eigenvalues and eigenvectors of unsymmetric matrices using a distributed memory multiprocessor. Technical Report ONRL/TM-10938, Oak Ridge National Laboratory, 1988.

[47] G. A. Geist, A. Lu, and E. Wachspress. Stabilized reduction of an arbitrary matrix to tridiagonal form. Technical Report ONRL/TM-11089, Oak Ridge National Laboratory, 1989.

[48] A. J. Geurts. A contribution to the theory of condition. *Num. Math.*, 39:85–96, 1982.

[49] D. Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1), 1991.

[50] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Num. Anal. (Series B)*, 2(2):205–224, 1965.

[51] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, 2nd edition, 1989.

[52] W. W. Hager. Condition estimators. *SIAM J. Sci. Stat. Comput.*, 5:311–316, 1984.

[53] D. J. Higham and N. J. Higham. Backward error and condition of structured linear systems. *SIAM J. Mat. Anal. Appl.*, 13:162–175, 1992.

[54] N. J. Higham. A survey of condition number estimation for triangular matrices. *SIAM Review*, 29:575–596, 1987.

[55] N. J. Higham. Algorithm 674: FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Soft.*, 14:381–396, 1988.

[56] N. J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Soft.*, 14:381–396, 1988.

[57] N. J. Higham. Experience with a matrix norm estimator. *SIAM J. Sci. Stat. Comput.*, 11:804–809, 1990.

[58] N.J. Higham. Iterative refinement enhances the stability of QR factorization methods for solving linear equations. *BIT*, 31:447–468, 1991.

[59] E. Jessup. A case against a divide and conquer approach to the nonsymmetric eigenproblem. Technical Report ONRL/TM-11903, Oak Ridge National Laboratory, 1991.

[60] W. Kahan. Paranoia. available from Netlib[35].

[61] W. Kahan. Accurate eigenvalues of a symmetric tridiagonal matrix. Computer Science Dept. Technical Report CS41, Stanford University, Stanford, CA, July 1966 (revised June 1968).

[62] W. Kahan. Analysis and refutation of the International Standard ISO/IEC for Language Compatible Arithmetic. SIGNUM Newsletter and SIGPLAN Notices, 1991.

[63] W. Kahan. How Cray's arithmetic hurts scientific computing. Presented to Cray User Group Meeting, Toronto, April 10, 1991.

[64] A. S. Krishnakumar and M. Morf. Eigenvalues of a symmetric tridiagonal matrix: a divide and conquer approach. *Numer. Math.*, 48:349–368, 1986.

[65] J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. Computer science department, Columbia University, Argonne, IL, March 1989. (to appear in SIMAX).

[66] U. Kulisch and W. Miranker, editors. *A new approach to scientific computing.* Academic Press, New York, 1983.

[67] H. T. Kung. Technical report, Computer Science Dept., Carnegie Mellon University, 1974. Also in [20].

[68] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *JACM*, 27(4):831–838, 1980.

[69] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

[70] T.-Y. Li and Z. Zeng. Homotopy-determinant algorithm for solving nonsymmetric eigenvalue problems. submitted to Math. Comp.

[71] C-C. Lin and E. Zmijewski. A parallel algorithm for computing the eigenvalues of an unsymmetric matrix on an SIMD mesh of processors. Department of Computer Science TRCS 91-15, University of California, Santa Barbara, CA, July 1991.

[72] S. Linnainmaa. Software for doubled-precision floating point computations. *ACM Trans. Math. Soft.*, 7:272–283, 1981.

[73] W. Mascarenhas. A note on Jacobi being more accurate than QR. Technical report, University of Minnesota, 1992. Submitted to SIAM J. Mat. Anal. Appl.

[74] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right hand sides. *Num. Math.*, 6:405–409, 1964.

[75] M.H.C. Paardekooper. A quadratically convergent parallel jacobi process for diagonally dominant matrices with distinct eigenvalues. *J. Comput. Appl. Math.*, 27:3–16, 1989.

[76] M. Payne and B. Wichmann. Information technology - programming languages - language compatible arithmetic. Project JTC1.22.28, ISO/IEC JTC1/SC22/WG11, 1 March 1991. First Committee Draft (Version 3.1).

[77] D. Priest. Algorithms for arbitrary precision floating point arithmetic. In P. Kornerup and D. Matula, editors, *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 132–145, Grenoble, France, June 26-28 1991. IEEE Computer Society Press.

[78] A. Sameh. On Jacobi and Jacobi-like algorithms for a parallel computer. *Math. Comp.*, 25:579–590, 1971.

[79] G. Shroff. A parallel algorithm for the eigenvalues and eigenvectors of a general complex matrix. *Numer. Math.*, 58:779–805, 1991.

[80] R. D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.*, 35:817–832, 1980.

[81] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines – EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1976.

[82] D. Sorensen and P. Tang. On the orthogonality of eigenvectors computed by divide-and-conquer techniques. *SIAM J. Num. Anal.*, 28(6):17572–1775, 1991.

[83] G. W. Stewart. A Jacobi-like algorithm for computing the Schur decomposition of a non-Hermitian matrix. *SIAM J. Sci. Stat. Comput.*, 6:853–864, 1985.

[84] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990.

[85] P. Swarztrauber. A parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix. *Math. Comp.*, 1992. submitted.

[86] P. Van Dooren. The computation of Kronecker's canonical form of a singular pencil. *Lin. Alg. Appl.*, 27:103–141, 1979.

[87] K. Veselić. On a class of Jacobi-like procedures for diagonalizing arbitrary real matrices. *Num. Math.*, 33:157–172, 1979.

[88] K. Veselić. A quadratically convergent Jacobi-like method for real matrices with complex conjugate eigenvalues. *Num. Math.*, 33:425–435, 1979.

[89] R. C. Ward. The QR algorithm and Hyman's method on vector computers. *Math. Comp.*, 30(33):132–142, January 1976.

[90] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, 1965.