LAPACK Working Note #4

Guidelines for the Design of Symmetric Eigenroutines, SVD and Iterative Refinement for Linear Systems

James Demmel, Jeremy Du Croz, Sven Hammarling and Dan Sorensen

March 1988

Abstract

This note summarizes the numerical and software issues which arise in designing the LAPACK subroutines for the symmetric eigenproblem, the singular value decomposition (SVD) and iterative refinement for linear systems. At the end of each chapter are a list of design questions for which we would like feedback from the user community.

Chapter 1

Guidelines for the Symmetric Eigenroutines in LAPACK

1.1 Introduction

This chapter discusses numerical and software design issues arising in the symmetric eigenroutines in LAPACK. Section 2 below summarizes the three algorithms available. Section 3 lists the criteria we will use to evaluate the algorithms. Section 4 lists the various computing environments and user options which will impact the choice of algorithm. Sections 5 to 8 make detailed comparisons of the three algorithms. Section 9 proposes an easy-to-use driver for the overall problem. Finally, section 10 lists design questions for which we would like feedback from the user community.

1.2 Available Algorithms

- QR and its variations - Here we include not only the standard explicit or implicit symmetric tridiagonal QR algorithm, but variations such as TQLRAT or PWK which are designed be go faster if no eigenvectors are desired [8, p. 164f]. This includes the possibility of using TQLRAT or PWK to compute all the eigenvalues followed by inverse iteration to compute selected eigenvectors. We will call this collection of methods QR for short.
- Divide and Conquer - This algorithm has been developed by Dongarra, Sorensen and Cuppen [12]. We will call it D&C for short.
- Bisection/Multisection - Here we include not just traditional bisection based on Sturm sequences followed by inverse iteration for the eigenvectors, but improvements based on multisection (in a parallel environment), using higher order zero finders (like ZEROIN) rather than just bisection, and reorthogonalization to guarantee orthogonality of eigenvectors corresponding to clustered eigenvalues. A prototype of such a code has been developed by Sameh, Lo and Philippe at Illinois [10]. We will call it B/M for short.

1.3 Comparison Criteria

- Accuracy (including special cases, like diagonal dominance)

- Susceptibility to Over/underflow
- Speed
- Storage

1.4 Options/Environments

- Original data dense or tridiagonal
- Serial or parallel algorithm
- Which eigenvalues, eigenvectors desired:
  - Some eigenvalues only
  - All eigenvalues only
  - Some eigenvalues and associated eigenvectors
  - All eigenvalues and eigenvectors

1.5 Accuracy Tradeoffs Among QR, D&C, B/M

The accuracy achievable depends on whether the initial data is tridiagonal or dense, and on special properties of the matrix. The usual perturbation theory says that all eigenvalues are determined by the data to within absolute accuracy $f(n)\epsilon \cdot \|A\|$, $n = dim(A)$ and $f(n)$ a modest function of n, provided each component of the initial data is known to that absolute accuracy. In other words, small *absolute* perturbations in the data cause small *absolute* perturbations in the eigenvalues. All three algorithms can compute the eigenvalues to this accuracy. Unless something special is known about the data, no more can be said. There are at least two special cases where the eigenvalues can be determined more accurately, because a stronger perturbation theorem is true: small *relative* perturbations in the data cause small *relative* perturbations in the eigenvalues.

- If A is tridiagonal with 0 diagonal, and all the data is known to within relative accuracy η, all the eigenvalues are determined to within relative accuracy $2n\eta$. This is equivalent to the SVD of a bidiagonal matrix. (See LAPACK Working Note #3 [13] and [9].)
- If A is tridiagonal and diagonally dominant (defined below), and all the data is known to within relative accuracy η, all the eigenvalues are determined to within relative accuracy of at most approximately $n\eta/(1-\gamma)$, where γ measures the diagonal dominance as follows: Suppose A has diagonal entries $a_1, ..., a_n$ and offdiagonal entries $b_1, ..., b_{n-1}$. Then $\gamma \equiv 2\max_i |b_i| \cdot |a_i \cdot a_{i+1}|^{-1/2}$. Note that this definition permits graded matrices which are not diagonally dominant in the more traditional sense. (See [9, 13].)

There are doubtless other cases where the eigenvalues (and eigenvectors) are determined more accurately by the data than the usual absolute bound $f(n)\epsilon \cdot \|A\|$. As long as an algorithm determines the eigenvalues with a small componentwise relative backward error (i.e. computes the exact eigenvalues of $A + \delta A$ with $|\delta A_{ij}| \leq \eta |A_{ij}|$), then if there is a relative perturbation theorem as above (i.e. small relative changes in the data cause small relative changes in the eigenvalues), the eigenvalues will be computed accurately.

1.5.1 Original Data Tridiagonal

- QR - If the tridiagonal has 0 diagonal, it could be changed into a bidiagonal SVD and the SVD QR used to get the eigenvalues to high relative accuracy, but this would be quite inconvenient to check for. Work is underway to evaluate QR and its variants (PWK) to see if they can compute eigenvalues accurately for diagonally dominant tridiagonals; this is a research question (we have no good idea as to what convergence criterion to use), so we should not count on QR for this problem. Also, QR does not guarantee small componentwise backward error.
- D&C - It appears difficult to guarantee high relative accuracy in any situation from the zero finder in D&C.
- B/M - B/M is definitely the best of the three algorithms for accuracy. It can always guarantee a tiny componentwise relative backward error independent of the data (modulo over/underflow), and so will compute the eigenvalues as accurately as they deserve.

1.5.2 Original Data Dense

Unless something special is known about the data, all that can be said is that the resulting tridiagonal is exactly similar to a matrix within a small absolute distance $f(n)\epsilon\|A\|$ of the original data A. Thus no (nonartificial) class of matrices is known where the eigenvalues are determined by the data to better than absolute accuracy. Therefore, each of QR, D&C and B/M has the same provable error properties: guaranteed absolute accuracy.

1.5.3 B/M and Monotonic Arithmetic

B/M is based on using a Sturm sequence to count the number of eigenvalues $n(z)$ less than z for any z. In exact arithmetic, $n(z)$ is a monotonic step function with unit increases at the eigenvalues. If the arithmetic is monotonic (eg. $a \geq b$ implies $fl(a+c) \geq fl(b+c)$, etc.), and if the inner loop of the Sturm sequence is implemented correctly, then the computed value of $n(z)$ is also monotonic. If we could assume $n(z)$ were monotonic, this might simplify the logic in an implementation of B/M; we do not yet know if this is important. If it does turn out to be a useful simplifying assumption, should we use it, and warn the user in the documentation?

1.6 Over/Underflow Susceptibility Tradeoffs Among QR, D&C, B/M

Here we deal with tridiagonal data only, since the reduction to tridiagonal form is the same for all three algorithms. Over/underflow is an issue because various researchers have reported underflow problems with the EISPACK routines recently. Briefly, since the current codes square and sometimes cube the initial data, they are susceptible to over/underflow when the data lies outside $[\lambda^{1/2}, \Lambda^{1/2}]$ or sometimes $[\lambda^{1/3}, \Lambda^{1/3}]$ (here λ is the underflow threshold and Λ is the overflow threshold). It appears that this can be avoided at the cost of a few more multiplies inside the inner loop (to avoid precomputing all the b_i^2), and changing the shift computation. The goal would be to make over/underflow impossible or comparable to roundoff for initial data much closer to the full range $[\lambda, \Lambda]$; some safety margins at the top and bottom are acceptable, but no constant fraction (i.e. half) of the exponent range could be excluded. Clearly, the shift should never overflow unless the largest eigenvalue of A itself is close to Λ, and it should never underflow unless the smallest eigenvalue of A is close to or less than λ.

- QR- The shift calculation needs to be changed to avoid squaring and sometimes cubing the data. The inner loop of standard implicit QR needs to be examined. It appears we can change

the inner loop of PWK (square root free QR) to make it satisfactorily robust with respect to over/underflow, but at the cost of reinserting a square-root. Is this acceptable? Another option is to scale the data based on the machine dependent constants λ and Λ before running the standard algorithm. Is it acceptable to do such a machine dependent scaling?

- D&C- The susceptibility to over/underflow of this algorithm is not known.
- B/M- If we avoid presquaring the offdiagonals b_i (and so add one multiply to the inner loop), the standard bisection algorithm can be made very robust. If we are willing to test and scale in the innerloop based on λ and Λ (machine dependent), it can be made ironclad; is this ok? Unfortunately, the faster version of the code which uses ZEROIN to find zeros of the determinant of the shifted matrix is extremely sensitive to over/underflow, and would not be reliable without scaling inside the inner loop, as is done to compute the determinant in the LINPACK routine SGEDI. Is there an alternative to finding zeros of the determinant (eg. finding zeros of the last pivot) which would eliminate the need for scaling? Or is scaling inside the inner loop acceptable?

1.7 Speed Tradeoffs Among QR, D&C and B/M

We only discuss tridiagonal data here, since the reduction to tridiagonal form is the same for all three. If the original data is dense, the time for reduction to tridiagonal form may overwhelm the differences between algorithms we discuss below. This is particularly likely if eigenvalues only are desired, since the cost of this is $O(n^2)$ whereas the reduction costs $O(n^3)$. Here the important environmental features and options are:

- Serial or parallel algorithm
- Which eigenvalues, eigenvectors desired:
  - Some eigenvalues only
  - All eigenvalues only
  - Some eigenvalues and associated eigenvectors
  - All eigenvalues and eigenvectors

1.7.1 Serial Algorithm

Some eigenvalues only

- QR- No consistent advantage can be taken of this case.
- D&C- No consistent advantage can be taken of this case.
- B/M- This is the method of choice, at least if a small enough fraction of the spectrum is desired. EISPACK says if 25% or fewer of the eigenvalues are desired, use B/M. Exact threshold between B/M and QR will be machine dependent.

All eigenvalues only

- QR- Probably fastest, especially with the PWK variant.
- D&C- Slowest, because it has to compute eigenvectors as well.
- B/M- Slower than QR, but not clear if cost is more than a factor of 2, or less (will be machine dependent).

Some eigenvalues and associated eigenvectors

Here it seems hard to rank the algorithms, because it depends strongly on what fraction of the spectrum is desired; the best choice will be machine dependent.

- QR- The algorithm computes all the eigenvalues and then uses inverse iteration with ultimate shifts for the desired eigenvectors. This might be fast depending on the fraction of the spectrum desired.
- D&C- Probably fastest if large enough fraction of eigenpairs desired.
- B/M- Fastest if small enough fraction of eigenpairs desired (use inverse iteration plus reorthogonalization for eigenvectors).

All eigenvalues and eigenvectors

- QR- Second fastest.
- D&C- Fastest.
- B/M- Slowest (hard to say how much slower than fastest).

1.7.2 Parallel Algorithm

Some eigenvalues only

- QR- No consistent advantage can be taken of this case.
- D&C- No consistent advantage can be taken of this case.
- B/M- This is the method of choice, at least if a small enough fraction of the spectrum is desired.

All eigenvalues only

- QR- Hard to parallelize, no advantage over serial case.
- D&C- Slowest, because it has to compute eigenvectors as well.
- B/M- Fastest.

Some eigenvalues and associated eigenvectors

Here it seems hard to rank the algorithms, because it depends strongly on what fraction of the spectrum is desired.

- QR- Slowest.
- D&C- Fastest if large enough fraction of eigenpairs desired.
- B/M- Fastest if small enough fraction of eigenpairs desired (use inverse iteration plus reorthogonalization for eigenvectors).

All eigenvalues and eigenvectors

- QR- Slowest.
- D&C- Perhaps fastest (testing needed).
- B/M- Perhaps fastest (testing needed).

1.8 Storage Tradeoffs Among QR, D&C, and B/M

Here the important environmental features and options are:

- Original Data Dense or Tridiagonal
- Which eigenvalues, eigenvectors desired:
  - Some eigenvalues only
  - All eigenvalues only
  - Some eigenvalues and associated eigenvectors
  - All eigenvalues and eigenvectors

The following conclusion are tentative, and could change as the implementation details are worked out.

1.8.1 Original Data Dense

Some eigenvalues only

All algorithms use $n^2 + O(n)$

All eigenvalues only

All algorithms use $n^2 + O(n)$

Some eigenvalues and associated eigenvectors

Let the number of eigenpairs be k.

- QR- $n^2 + kn+$ lower order (eigenvalues alone followed by ultimate shifts for accumulation).
- D&C- $1.5n^2+$ lower order; there is no savings for k eigenpairs only. The coefficient 1.5 is difficult to attain; 2 is easier.
- B/M- $n^2 + kn+$ lower order.

All eigenvalues and eigenvectors

- QR- n^2+ lower order.
- D&C- $1.5n^2+$ lower order. As before, the coefficient 1.5 is hard to attain; 2 is easier.
- B/M- $2n^2+$ lower order ($1.5n^2$ if packed?).

1.8.2 Original Data Tridiagonal

Some eigenvalues only

$O(n)$ for all.

All eigenvalues only

$O(n)$ for all.

Some eigenvalues and associated eigenvectors

- QR- $kn+$ lower order (all eigenvalues, then inverse iteration).
- D&C- n^2+ lower order (no savings for k eigenvectors only).
- B/M- $kn+$ lower order.

All eigenvalues and eigenvectors

n^2+ lower order for all.

1.9 Recommendations for Easy-To-Use Driver

We propose the following underlying philosophy for the design of easy to use drivers: pick the most accurate routine as long as the performance or storage penalty is not too large. In this case, if the original data is tridiagonal, we would always use B/M. The largest performance penalty is when computing all eigenvalues and eigenvectors using the serial algorithm, and it is not yet clear whether the penalty is more or less than a factor of 2. If the original data is dense, and only eigenvalues are desired, we again use B/M, and note that there is essentially no performance penalty because the cost of the reduction to tridiagonal form ($O(n^3)$) overwhelms the cost of finding the eigenvalues alone ($O(n^2)$). When eigenvectors are desired as well, the cost of the reduction is no longer overwhelming and one is tempted to use whatever algorithm is fastest, because the reduction to tridiagonal form destroys whatever provable accuracy advantage B/M had. Or is this argument too much like the one which says "computing the sine of large arguments accurately is unimportant because large arguments are probably inexact, so any value for the sine is good enough"? In other words, should we always use the most accurate tridiagonal eigenroutine even though the reduction to tridiagonal form probably destroys the relative accuracy of tiny eigenvalues?

1.10 Questions for the Community

This section summarizes the questions raised in previous sections.

1. Is our proposed philosophy for the design of easy-to-use drivers appropriate: use the most accurate routine as long as the performance penalty is less than a factor of 2?
2. Should we incorporate possibly machine dependent scaling to avoid over/underflow, especially in the easy-to-use code?
3. Should we assume that arithmetic is monotonic if that turns out to be convenient in the multisection code?
4. Has the cost of square root decreased sufficiently compared to division that it is no longer necessary to have a square root free QR algorithm for the sake of speed?

5. In B/M, is there an alternative to finding the zeros of the determinant which is less susceptible to over/underflow?

Chapter 2

Guidelines for the SVD Routines in LAPACK

2.1 Introduction

This note discusses numerical and software design issues arising in the singular value decomposition eigenroutines in LAPACK. Many of the issues are similar to the ones arising in the symmetric eigenproblem. Section 2 below summarizes the three algorithms available. Section 3 discusses the criteria we will use to evaluate the algorithms. Section 4 discusses the various computing environments and user options which will impact the choice of algorithm. Sections 5 to 8 make detailed comparisons of the three algorithms. Section 9 proposes an easy-to-use driver for the overall problem. Finally, section 10 lists design questions for which we would like feedback from the user community.

2.2 Available Algorithms

- QR and its variations - Here we refer to the variation of QR discussed in LAPACK Working Note #3 [13], which is a new algorithm guaranteed to find all the singular values of a bidiagonal matrix to full working precision independent of their magnitudes. This includes the possibility of first finding the singular values and then using inverse iteration to find selected singular vectors. We call this algorithm QR for short.

- Divide and Conquer - This algorithm is similar to the Dongarra/Sorensen/Cuppen algorithm for the symmetric tridiagonal eigenproblem. It has been developed by Jessup and Sorensen [11]. We call it D&C for short.

- Bisection/Multisection - This algorithm is similar to the algorithm for the symmetric tridiagonal eigenproblem developed by Sameh, Lo and Philippe [10]. It uses the fact the the bidiagonal singular value problem can be converted into a symmetric tridiagonal eigenproblem with zero diagonal.

2.3 Comparison Criteria

- Accuracy
- Susceptibility to Over/underflow
- Speed
- Storage

2.4 Options/Environments

- Original data dense or bidiagonal
- Serial or parallel algorithm
- Which singular values, singular vectors desired:
  - Some singular values only
  - All singular values only
  - Some singular values and associated (left and/or right) singular vectors
  - All singular values and (left and/or right) singular vectors ("decomposition" or "factorization", as in LINPACK)

2.5 Accuracy Tradeoffs Among QR, D&C, B/M

The accuracy achievable depends on whether the initial data is bidiagonal or dense. The usual perturbation theory says that all singular values are determined by the data to within absolute accuracy $f(n)\epsilon \cdot \|A\|$, $n = dim(A)$ and $f(n)$ a modest function of n, provided each component of the initial data is known to that absolute accuracy. In other words, small *absolute* perturbations in the data cause small *absolute* perturbations in the singular values. All three algorithms can compute the singular values to this accuracy. If the data is bidiagonal, more can be done because a stronger perturbation theorem is true: small *relative* perturbations in the data cause small *relative* perturbations in the singular values. As for the singular vectors, the usual perturbation bound is of the form $f(n)\epsilon/absolute\_gap$, where $absolute\_gap$ is the absolute difference $min|\sigma_i - \sigma_{i\pm1}|/\sigma_1$ between the corresponding singular value $sigma_i$ and its nearest neighbor (scaled by the matrix norm $sigma_1$); any of the algorithms can compute the singular vectors to this accuracy from dense or bidiagonal data. When the data is bidiagonal, the singular vectors appear to be determined more accurately: we conjecture that the perturbation bound may be improved to $f(n)\epsilon/relative\_gap$, where $relative\_gap$ is the relative distance $min|\sigma_i - \sigma_{i\pm1}|/\sigma_i$ between the corresponding singular value $sigma_i$ and its nearest neighbor. It appears the variant of QR discussed below can attain this accuracy (we have an outline of a proof; see section 10 of LAPACK working note #3). For example, if a 3 by 3 matrix has singular values $1, 2 \cdot 10^{-100}, 10^{-100}$, then the conventional SVD computes the singular vectors corresponding to the smaller two singular values with relative precision on the order $\epsilon 10^{100}$, i.e. probably none. The new algorithm, however, appears to compute them all to precision ϵ. For futher discussion see [13].

2.5.1 Original Data Bidiagonal

- QR- The variant of QR in LAPACK working note #3 ("implicit zero-shift QR") can compute all the singular values to full relative precision. We conjecture that with a somewhat more conservative stopping criteria (which would make the algorithm somewhat slower - how much we don't know) the singular vectors can also be computed very accurately as described above.
- D&C- It appears difficult to guarantee high relative accuracy in any situation from the zero finder in D&C.
- B/M- B/M can also compute all the singular values to guaranteed high relative accuracy. It is unclear how accurately we can compute the singular vectors via inverse iteration (certainly to within the conventional error bound, but perhaps not to the conjectured higher one).

2.5.2 Original Data Dense

Unless something special is known about the data, all that can be said is that the resulting bidiagonal is exactly orthogonally equivalent to a matrix within a small absolute distance $f(n)\epsilon\|A\|$ of the original data A. Thus no (nonartificial) class of matrices is known where the singular values are determined by the data to better than absolute accuracy. Therefore, each of QR, D&C and B/M has the same provable error properties: guaranteed absolute accuracy.

2.5.3 B/M and Monotonic Arithmetic

This issue is identical to the case of the symmetric tridiagonal eigenproblem.

2.6 Over/Underflow Susceptibility Tradeoffs Among QR, D&C, B/M

This issue is identical to the case of the symmetric tridiagonal eigenproblem.

2.7 Speed Tradeoffs Among QR, D&C and B/M

We only discuss bidiagonal data here, since the reduction to bidiagonal form is the same for all three. If the original data is dense, the time for reduction to bidiagonal form may overwhelm the differences between algorithms we discuss below. This is particularly likely if singular values only are desired, since the cost of this is $O(n^2)$ whereas the reduction costs $O(n^3)$. Here the important environmental features and options are:

- Serial or parallel algorithm
- Which singular values, singular vectors desired:
  - Some singular values only
  - All singular values only
  - Some singular values and associated singular vectors
  - All singular values and singular vectors

The analysis is identical to the case of the symmetric tridiagonal eigenproblem.

2.8 Storage Tradeoffs Among QR, D&C, and B/M

Here the important environmental features and options are:

- Original Data Dense or Bidiagonal
- Which singular values, singular vectors desired:
  - Some singular values only
  - All singular values only
  - Some singular values and associated singular vectors
  - All singular values and singular vectors

In all cases the analysis is similar to the case of the symmetric tridiagonal eigenproblem, except that separate storage is needed for right and left transformations and singular vectors. This is a tentative conclusion that might change as the details of the implementation are worked out.

2.9 Recommendations for Easy-To-Use Driver

The underlying philosophy is to pick the most accurate routine as long as the performance or storage penalty is not too large. If the algorithm is serial and all or most singular values (and possibly vectors) are needed, use QR. If singular vectors are desired, use the more conservative (and somewhat slower) stopping criterion discussed in section 10 of LAPACK working note #3. If the algorithm is serial and only a few singular values (and possibly vectors) are needed, use B/M. If the algorithm is parallel, use B/M. This should combine nearly highest speed and accuracy in all cases.

2.10 Questions for the Community

The questions pertinent to the SVD are essentially the same as for the symmetric eigenproblem of the last chapter:

1. Is our proposed philosophy for the design of easy-to-use drivers appropriate: use the most accurate routine as long as the performance penalty is less than a factor of 2?
2. Should we incorporate possibly machine dependent scaling to avoid over/underflow, especially in the easy-to-use code?
3. Should we assume that arithmetic is monotonic if that turns out to be convenient in the multisection code?
4. In B/M, is there an alternative to finding the zeros of the determinant which is less susceptible to over/underflow?

Chapter 3

Guidelines for Iterative Refinement and Condition Estimation in LAPACK

3.1 Background

There are two general techniques for computing error bounds for solutions of linear systems. The first is based on computing the backward error ω, estimating the condition number κ, and multiplying them to get an error bound $\omega \cdot \kappa$. The second technique is based on iterative refinement with extended precision (usually double precision) residual calculations, and examining the convergence rate.

3.1.1 Techniques based on backward error and condition estimation

This method is the most flexible of the two, and described in some detail in [1]. We outline the idea here. Given an approximate solution $\hat{x}$ to the linear system $Ax = b$, we seek the *backward error*, i.e. the size ω of the smallest δA and δb such that

$$(A + \delta A)\hat{x} = b + \delta b.$$

To find ω, we need to specify norms for δA and δb; ω will depend on this choice of norm. Here we will be as flexible as possible, and choose a different scaling for each entry of δA and δb as follows: Let E be a nonnegative matrix and f a nonnegative vector. Then the norm of δA and δb with respect to E and f is denoted $\|(\delta A, \delta b)\|_{E,f}$ and defined as the smallest ω such that

$$|\delta A_{ij}| \leq \omega E_{ij} \quad \text{and} \quad |\delta b_i| \leq \omega f_i \quad \text{for all } i, j$$

For example, if $E_{ij} = \|A\|_\infty$ and $f_i = \|b\|_\infty$, then this corresponds essentially to the usual normwise or Wilkinson backward error. If we choose $E_{ij} = |A_{ij}|$ and $f_i = |b_i|$, then this corresponds to a *componentwise relative backward error* discussed in [4,5,6]. This backward error maintains the sparsity structure of A and b, since if $A_{ij} = 0$, δA_{ij} must equal zero, and similarly for δb_i. Thus it is a significantly more stringent backward error measure than the normwise one.

It turns out that it costs just two matrix-vector multiplies to compute ω:

$$\omega = \max_i \frac{|A\hat{x} - b|_i}{(E|\hat{x}| + f)_i}$$

(here $|\hat{x}|$ denotes the vector of absolute entries of $\hat{x}$; similarly, $|A|$ denote the matrix of absolute entries of A). Thus it is quite cheap to compute ω for an arbitrary backward error, making it suitable either for error estimation or as a stopping criterion for iterative refinement.

The reason the componentwise relative backward error is interesting is the following theorem of Skeel [5]:

Theorem: If A is not too ill-conditioned, and if the components of the vector $|A| \cdot |x|$ do not vary too much in magnitude, then one step of iterative refinement guarantees that the componentwise relative backward error is on the order of machine precision. This is true even if the residuals are computed in single precision.

Note that this theorem violates the conventional wisdom that it is not worth doing iterative refinement unless the residuals are computed to higher than single precision. The assumption about the components of $|A| \cdot |x|$ may be violated if both A and b are sparse, so in practice we must sacrifice the sparsity structure of δb by occasionally permitting f_i to be larger than $|b_i|$. However in practice we can always guarantee that δA has the same sparsity structure as A; see [1] for details. Thus, it is quite worthwhile to do a single step of iterative refinement with single precision residuals.

Corresponding to the backward error ω which depends on E and f is a condition number which also depends on E and f. Suppose $Ax = b$ and $(A + \delta A)(x + \delta x) = b + \delta b$. We define the condition number of the system $Ax = b$ with respect to E and f as

$$\kappa_{E,f}(A, b) \equiv \limsup_{\substack{\delta A \to 0 \\ \delta b \to 0}} \frac{\|\delta x\|_\infty / \|x\|_\infty}{\|(\delta A, \delta b)\|_{E,f}} = \frac{\| \, |A^{-1}| \cdot (E \cdot |x| + f)\|_\infty}{\|x\|_\infty} \quad .$$

Since the true solution x is usually unknown, we must approximate this condition number somehow. One way is to substitute the computed solution $\hat{x}$ for x. The other way is to use an x which maximizes the condition number. For the normwise backward error, this yields the usual condition number $\kappa = \|A\|_\infty \cdot \|A^{-1}\|_\infty$, and for the componentwise relative backward error, this yields $\kappa = \| \, |A^{-1}| \cdot |A| \, \|_\infty$. Either way, an approximate error bound is obtained by multiplying the backward error ω by the condition number κ:

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \leq \kappa \cdot \omega$$

The advantage of using $\| \, |A^{-1}| \cdot |A| \, \|_\infty$ over the more conventional $\|A\|_\infty \cdot \|A^{-1}\|_\infty$ is that it is no larger, and sometimes much smaller, especially if A is badly row-scaled (note that $\| \, |A^{-1}| \cdot |A| \, \|_\infty$ is independent of the row-scaling of A). In other words, one step of iterative refinement tends to correct poor row scaling [6]. Therefore, the error estimate $\kappa \cdot \omega$ may be much smaller using the componentwise relative backward error and condition number after one step of iterative refinement than using the normwise backward error and condition number [1].

To be useful, it is necessary to be able to estimate the condition number cheaply. It turns out to be inexpensive to get reliable estimates of $\kappa_{E,F}(A, b)$ for any E and f, based on an estimator in [2,3] for estimating the one-norm or infinity-norm of a matrix B given the ability to quickly form Bx or $B^T x$ for any vector x; when $B = A^{-1}$ and the LU factors of A are available, this is indeed inexpensive. The idea is as follows:

$$\| \, |A^{-1}|(E|\hat{x}| + f) \, \|_\infty = \| \, |A^{-1}| g \, \|_\infty$$

where $g = E|\hat{x}| + f$ can be computed with a single matrix-vector multiply. Similarly

$$\| \, |A^{-1}| \cdot |A| \, \|_\infty = \| \, |A^{-1}| \cdot |A| \cdot e \, \|_\infty = \| \, |A^{-1}| \cdot g \, \|_\infty$$

where e is the column vector of all ones and $g = |A|e$ can be computed with a single matrix-vector multiply. In any event, we need to be able to estimate $\| \, |A^{-1}|g \, \|_\infty$ where g is a nonnegative vector. We can do this using the estimator in [2,3] as follows: Let $G = diag(g_1, ..., g_n)$. Then $g = Ge$ and

$$\| \, |A^{-1}| \cdot g \, \|_\infty = \| \, |A^{-1}| \cdot Ge \, \|_\infty = \| \, |A^{-1}| \cdot G \, \|_\infty = \| \, |A^{-1}G| \, \|_\infty = \| \, A^{-1}G \, \|_\infty .$$

Thus, to use [2,3] we need to be able to multiply by $A^{-1}G$ (and its transpose) which is easy since G is diagonal and we have the LU factors of A.

3.1.2 Techniques based on Iterative Refinement with Extended Precision Residuals

This is a standard technique suggested by Wilkinson. If the residuals are computed to double precision, then each step of iterative refinement should increase the accuracy of the computed solution by a factor of approximately $\epsilon \cdot \|A\| \cdot \|A^{-1}\|$. Letting x^i denote the solution after i steps of iterative refinement, this implies $\|x^i - x^{i-1}\|/(\epsilon\|x^i\|)$ should be an estimate of $\|A\| \cdot \|A^{-1}\|$, and that after x^i stops changing, it has converged to the correct solution. Unfortunately, this is not always reliable because iterative refinement can appear to converge even though the solution is completely wrong if A is sufficiently ill-conditioned. Also, the estimate of $\|A\| \cdot \|A^{-1}\|$ may be too high or too low. Indeed, in [7] Wilkinson recommends against using this method without an independent condition estimator. Nonetheless, iterative refinement with extended precision residuals is an important technique for improving the accuracy of a computed solution.

3.2 Design Issues for Iterative Refinement and Condition Estimation in LAPACK

3.2.1 Equilibration

A common technique to improve the conditioning of A is to scale its rows and/or columns by pre- and/or postmultiplying A by nonsingular diagonal matrices. Single sided scaling (i.e. replacing A by DA or AD with D diagonal and nonsingular) to make the rows (or columns) of A have equal norm can be shown to reduce the condition number (with respect to the 2-norm) to within a factor of $n^{1/2}$ of the minimal over all diagonal D. If A is symmetric positive definite, symmetric scaling DAD to make the diagonals of A all equal reduces the condition number to within a factor of n of minimal. It is not as well understood how to do two-sided scaling D_1AD_2; in general current algorithms attempt to make the rows and columns of D_1AD_2 have nearly equal norms. No such code currently exists in LINPACK; should one be added? Note that one step of iterative refinement with single precision residuals tends to automatically row-scale in a nearly optimal way.

3.2.2 Extended Precision Residual Accumulation

There are several difficulties with incorporating extended precision residual accumulation in a library. The first is that twice working precision is not available in a machine independent way, or sometimes not available at all: mixed precision BLAS implementations are not universally available, nor are quadruple and double complex arithmetic. Second, efficient residual computation would require a work vector to store the residual (in order to use a column-oriented algorithm); thus if both single and double precision residual accumulation are to be provided, either different subroutine names are required, or else both single and double precision work vectors would have to be passed, one of which would be ignored. The first solution is difficult because the limitation of 6 character subroutine names already makes naming difficult, and the second solution makes the calling sequence more complicated. Finally, as discussed in section 1.2, this is not a reliable method for error estimation

in the absence of an independent condition estimator. Therefore we propose not to include iterative refinement with extended precision residual accumulation as a standard part of LAPACK, although we will indicate how it could be written in terms of other subroutines in the library.

3.2.3 Easy-to-use Driver

There are two obvious possibilities for an easy to use driver, one corresponding to the choice of Wilkinson or normwise backward error, and the other corresponding to componentwise relative backward error. Either one would work as follows:

1. Do iterative refinement with a convergence criterion such as:

   Repeat until $\omega_i < n\epsilon$ or $\omega_i/\omega_{i-1} > .5$.

   In other words, we perform iterative refinement until the backward error is less than $n\epsilon$ or no longer decreases by at least a factor of 2. Variations on this include changing $n\epsilon$ to ϵ or some other value (as in [1]), or adding a maximum iteration count (as it stands, the number of iterations could be as large as the number of bits in the fraction of the floating point format, although this is highly unlikely).

2. Estimate the condition number κ. Actually, we would report $1/\kappa$ to the user, since the reciprocal is zero (rather than infinity) for an exactly singular matrix.

3. Compute the relative error bound $\kappa \cdot \omega$ and return it to the user. If this would overflow (or just be greater than 1), return 1. Also return the final residual vector to the user.

The major question is which measure of backward error should be used. The normwise measure is most robust in that it is essentially guaranteed to be small ("Gaussian elimination produces a small residual") even without iterative refinement. But this means iterative refinement would seldom be done, and even if it were the possible benefits would not be reflected in the condition number and error bound returned to the user (implicitly improved row scaling and smaller condition number, a tiny componentwise relative backward error). Alternatively, the componentwise relative backward error would reflect these benefits, but if A and b are sparse then ω may never be small, even if the solution is quite accurate.

A compromise would be to use $E = |A|$ (componentwise relative error in A) and $f_i = \|b\|$ (normwise error in b). This has worked well in practice on difficult sparse problems [1], although it does not always give the tightest error bound. One somewhat unpleasant feature of this backward error measure is that it is not row-scaling independent. (Another alternative, where f_i is chosen dynamically depending on the computation, is discussed in [1]).

A final possibility is to provide all three backward error measures and to make the user choose one of them; this is probably too complicated for the easy-to-use driver.

Presuming we choose exactly one backward error measure to be used by the easy-to-use driver, we propose the following calling sequence for it:

```
      SGESVE(TRANS, N, A, LDA, AF, LDAF, IPIV, B, X, R, E, RCOND, INFO)
      CHARACTER*1 TRANS
      INTEGER N, LDA, LDFA, IPIV(*), INFO
      REAL A(LDA,*), AF(LDFA,*), B(*), X(*), E, RCOND
C
C     ARGUMENTS
C       TRANS - CHARACTER*1
C               SPECIFIES FORM OF EQUATION TO SOLVE
C               IF TRANS = 'N' OR 'n', SOLVE A*X=B
```

```
C               IF TRANS = 'T' OR 't' OR 'C' OR 'c', SOLVE A'*X=B
C               UNCHANGED ON EXIT
C       N     - INTEGER - NUMBER OF ROWS AND COLUMNS OF A AND AF,
C               NUMBER OF ROWS OF B, R AND X
C               N MUST BE AT LEAST 0, UNCHANGED ON EXIT
C       A     - REAL ARRAY OF DIMENSION (LDA,*)
C               COEFFICIENT ARRAY, UNCHANGED ON OUTPUT
C       LDA   - INTEGER - LEADING DIMENSION OF A, UNCHANGED ON EXIT,
C               MUST BE AT LEAST MAX(1,N)
C       AF    - REAL ARRAY OF DIMENSION (LDAF,*)
C               CONTAINS LU FACTORIZATION OF A ON EXIT
C       LDAF  - INTEGER - LEADING DIMENSION OF AF, UNCHANGED ON EXIT
C       IPIV  - INTEGER ARRAY OF DIMENSION (*), NEEDS TO BE
C               AT LEAST MAX(1,N), CONTAINS PIVOT INFORMATION FOR
C               LU FACTORIZATION ON EXIT
C       B     - REAL ARRAY OF DIMENSION (N), CONTAINS RIGHT HAND SIDE,
C               UNCHANGED ON EXIT
C       X     - REAL ARRAY OF DIMENSION (N), CONTAINS SOLUTION ON EXIT
C       R     - REAL ARRAY OF DIMENSION (N), CONTAINS RESIDUAL A*X-B
C               ON EXIT
C       E     - REAL - CONTAINS RELATIVE ERROR BOUND FOR X ON EXIT
C       RCOND - REAL - CONTAINS RECIPROCAL OF ESTIMATED CONDITION NUMBER
C               OF A ON EXIT
C       INFO  - INTEGER -
C               IF INFO=0 ON EXIT, NORMAL TERMINATION
C               IF INFO .GT. 0 ON EXIT, POINTS TO FIRST ZERO PIVOT
C               IN U
C
```

There is also a version for multiple right hand sides:

```
      SGESME(TRANS, N, NRHS, A, LDA, AF, LDFA, IPIV, B, LDB,
     +       X, LDX, R, LDR, E, RCOND, INFO)
C     E - REAL ARRAY OF DIMENSION (NRHS), ON EXIT CONTAINS RELATIVE
C         ERROR BOUND FOR EACH RIGHT HAND SIDE
```

with the extra parameters LDB, LDX and LDR for the leading dimensions of B, X and R, and NRHS for the number of right hand sides (columns of B, X, and R).

3.2.4 User Supplied Measure of Uncertainly in the Data

Many users know or can estimate the accuracy ω_{user} to which their data is known. In such cases a more relevant error bound than $\omega \cdot \kappa$ (ω is the backward error determined by the code) would be $\omega_{user} \cdot \kappa$. Should the easy-to-use driver have an argument for ω_{user} and return the product $\omega_{user} \cdot \kappa$ in another argument? If so, how should ω_{user} be measured (normwise, componentwise relative, other)?

3.2.5 Test for Singularity During Triangular Factorization

We propose to have no test for singularity during triangular factorization, completing the factorization even if A has a completely zero column or row. The resulting factorization may still be of use to the user, even if it is not useful for iterative refinement.

3.3 Questions for the Community

This section summarizes the design questions raised in the previous sections.

1. Should an equilibration routine be included in the library? Which options should it allow (one-sided left or right, two-sided, symmetric)?
2. Should iterative refinement with double (or extended) precision residual calculation be left to the user?
3. Which backward error measure should be used in the easy-to-use driver (normwise, componentwise relative, a compromise between the two, user's choice of one of the three, other)?
4. Should the easy-to-use driver accept a user-supplied measure ω_{user} of the uncertainty in the data and compute an error bound $\omega_{user} \cdot \kappa$ based on it rather than the backward error ω of the algorithm? If so, with respect to which backward error measure should ω_{user} be measured?
5. Is it appropriate not to test for singularity during the factorization?

Chapter 4

References

1. M. Arioli, J. Demmel, I. Duff, "Solving Sparse Linear Systems with Sparse Backward Error," Harwell Laboratory, Computer Science and Systems Division, Report CSS 214, March 1988

2. W. Hager, "Condition Estimators," SIAM J. Sci. Stat. Comp. 5, (1984), p. 311-316

3. N. Higham, "Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation," Numerical Analysis Report No. 135, U. of Manchester, England, 1987

4. W. Oettli, W. Prager, "Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides," Numer. Math. 6, (1964) p. 405-409

5. R. Skeel, "Iterative refinement implies numerical stability for Gaussian elimination," Math. Comp. 35, (1980) p. 817-832

6. R. Skeel, "Scaling for numerical stability in Gaussian elimination," J. ACM 26, (1979) p. 494-526

7. J. Wilkinson, "Rounding Errors in Algebraic Processes," Prentice Hall, 1964

8. B. Parlett, "The Symmetric Eigenproblem," Prentice-Hall, 1980

9. W. Kahan, "Accurate eigenvalues of a symmetric tridiagonal matrix," Technical Report No. CS41, Computer Science Dept., Stanford University, July 22 1966 (revised June 1968)

10. S-S. Lo, B. Phillipe, A. Sameh, "A multiprocessor algorithm for the symmetric tridiagonal eigenproblem," SIAM J. Sci. Stat. Comp., Vol. 8, No. 2, March 1987, pp s155-165

11. E. R. Jessup and D. C. Sorsensen, "A Parallel Algorithm for Computing the Singular Value Decomposition of a Matrix," Technical Memorandum No. 102, Mathematics and Computer Science Division, Argonne National Lab, December 1987

12. J. Dongarra and D. Sorensen, "A fully parallel algorithm for the symmetric eigenproblem," SIAM J. Sci. Stat. Comp. 8, pp. 139-154, 1987

13. J. Demmel and W. Kahan, "LAPACK Working Note #3: Computing Small Singular Values of Bidiagonal Matrices With Guaranteed Relative Accuracy," Mathematics and Computer Science Division, Argonne National Lab, March 1988