Current inverse iteration software can fail

Inderjit S. Dhillon *
IBM Almaden Research Center
San Jose, California

February 15, 1998

Abstract

Inverse Iteration is widely used to compute the eigenvectors of a matrix once accurate eigenvalues are known. We discuss various issues involved in any implementation of inverse iteration for real, symmetric matrices. Current implementations resort to reorthogonalization when eigenvalues agree to more than three digits relative to the norm. Such reorthogonalization can have unexpected consequences. Indeed, as we show in this paper, the implementations in EISPACK [18] and LAPACK [1] may fail. We illustrate with both theoretical and empirical failures.

Keywords : Inverse iteration, symmetric, tridiagonal matrix, eigenvalues, eigenvectors.

AMS subject classification : 15A18, 65F15, 65F25.

1 Introduction

Given an eigenvalue λ of the matrix A, a corresponding eigenvector is defined as a non-zero solution of the homogeneous system

$$(A - \lambda I)v = 0.$$

However in a computer implementation we can only expect, in general, to have an approximation σ to λ. In such a case, we may attempt to compute an approximation to the eigenvector by the method of inverse iteration (see (3.3) below).

The inverse iteration process is widely used to find the eigenvectors of a symmetric tridiagonal matrix T. Earlier fears about loss of accuracy due to the near singularity of $T - \sigma I$ were allayed in [17]. Inverse iteration can easily deliver a vector $\hat{v}$ that has a small residual, i.e. small $\|(T - \sigma I)\hat{v}\|$, whenever σ is close to λ. However a small residual does not guarantee orthogonality when eigenvalues are close together. A simple and commonly used "remedy" for clusters of eigenvalues is to orthogonalize each approximate eigenvector, as soon as it is computed, against any eigenvectors

*This research was supported, while the author was at the University of California, Berkeley, in part by DARPA Contract No. DAAL03-91-C-0047 through a subcontract with the University of Tennessee, DOE Contract No. DOE-W-31-109-Eng-38 through a subcontract with Argonne National Laboratory, by DOE Grant No. DE-FG03-94ER25219, NSF Grant Nos. ASC-9313958 and CDA-9401156, and DOE Contract DE-AC06-76RLO 1830 through the Environmental Molecular Sciences construction project at Pacific Northwest National Laboratory (PNNL). The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

already in the cluster. Unfortunately, this strategy is not foolproof as recognized by Wilkinson [20, p.344] :

> 'Inverse Iteration gives a very satisfactory solution to the problem as far as reasonably well-separated eigenvalues are concerned. The problem of determining reliably full digital information in the subspace spanned by eigenvectors corresponding to coincident or pathologically close eigenvalues has never been satisfactorily solved'.

In this paper, we list the various issues involved in implementing inverse iteration. We discuss the plausible choices that may be made at each step, illustrating with the choices made in EISPACK [18] and LAPACK [1].

The task of computing eigenvectors has been extensively studied since the 1960s and there is considerable inverse iteration software in the public domain that is available for use. One would expect that software for this well-defined and seemingly "simple" task would come with proofs of correctness. These proofs could be for exact arithmetic (with an approximate eigenvalue) or for finite precision arithmetic. The effort to construct the proof should bring bugs in the software to light. However, as this study reveals, most existing inverse iteration software can fail, without documenting how failure can occur. Indeed a user gets the impression that the method is reliable and the relevant literature fosters that impression. Although EISPACK and LAPACK implementations work well in most cases, we have found instances where they deliver unnecessarily inaccurate answers and we explain the mechanism in this paper.

It is not the goal of this paper to suggest or investigate alternative strategies for the implementation of inverse iteration. In this paper, we want to carry out the intellectual exercise of examining difficulties faced by high quality software for this task. This effort is worthwhile since it reveals somewhat surprising errors in the EISPACK and LAPACK implementations. As far as the author knows, most of these instances of error have not previously appeared in literature. We feel that similar endeavors would uncover more pitfalls in existing numerical software, and lead to more reliable software.

The outline of the paper is as follows. Section 2 lists the goals of any floating point implementation of inverse iteration while in Section 3, we present and then discuss various issues that any such implementation must address. In Section 4 we examine in detail how these issues have been handled in existing EISPACK and LAPACK implementations. Our examination enables us to discover examples where these routines deliver inaccurate results. Finally in Section 5, we indicate the aspects of inverse iteration that can be satisfactorily solved at the present, and mention alternate approaches to the unresolved problems.

A word about notation. Unless explicitly mentioned, all matrices are assumed to be of order n. We will denote eigenvalues by $\lambda_1, \lambda_2, \ldots, \lambda_n$ and the corresponding eigenvectors by $v_1, v_2, \ldots, v_n$. The computed value of a quantity x will usually be denoted by $\hat{x}$, e.g., $\hat{\lambda}_i$ denotes the computed value of λ_i. The ith component of the vector x will be denoted by $x(i)$, while the (i, j) element of matrix A will be denoted by either A_{ij} or $A(i, j)$. Our usage should be clear from the context. Unless explicitly mentioned, $\|\cdot\|$ is the spectral norm, $\|x\|_2 = \sqrt{x^T x}$. In many of our expositions, we will abuse the "big oh" notation and instead of a limiting process, $O(x)$ will informally be used as a synonym for "of the order of magnitude of" x. The symbol ε will usually denote the machine precision.

Throughout the paper, A will denote a dense symmetric matrix while T will stand for a symmetric tridiagonal. We will use A and T interchangeably during the paper, and this reflects the

fact that while all the concepts are valid for real and symmetric matrices, inverse iteration software is typically written only for tridiagonals for purposes of efficiency.

2 Goals in finite precision

What are the proper requirements for any pair $(\hat{\lambda}_i, \hat{v}_i)$ to be accepted as a satisfactory approximate eigenpair for a given (machine representable) real and symmetric matrix A? It is customary to judge by the norm of the residual vector,

$$r_i := A\hat{v}_i - \hat{v}_i\hat{\lambda}_i, \quad \|\hat{v}_i\| \approx 1.$$

The standard criterion is

$$\|r_i\| \leq O(n\varepsilon\|A\|). \tag{2.1}$$

For theory we use the spectral norm but in practice simpler norms are preferred. The presence of n in the test merits further discussion but that is not our concern here.

What complicates the task considerably is a second requirement concerning mutual orthogonality of eigenvectors. The customary criterion is

$$|\hat{v}_i^T\hat{v}_j| \ \leq \ O(n\varepsilon), \qquad \|\hat{v}_i\|, \|\hat{v}_j\| \approx 1, \qquad i \neq j, \tag{2.2}$$

for approximations $\hat{v}_i$ and $\hat{v}_j$ to eigenvectors of A. We accept (2.2) without further discussion.

Note that in (2.1) and (2.2), the factor ε is generally taken to be the machine precision but may be larger to reflect the level of accuracy desired by the user.

3 Issues in Inverse Iteration

Inverse Iteration is a method to find an eigenvector when an approximate eigenvalue is known :

$$v^{(0)} = b, \ \ (A - \sigma I)v^{(i+1)} = \tau^{(i)}v^{(i)}, \quad i = 0, 1, 2, \ldots. \tag{3.3}$$

Here b is the starting vector. Usually $\|v^{(i)}\| = 1$ and $\tau^{(i)}$ is a scalar chosen to try and make $\|v^{(i+1)}\| \approx 1$. We now list the key issues that arise in a computer implementation.

I. **Choice of shift.** What shift σ should be chosen when doing the inverse iteration steps of (3.3)? Should σ always equal the best possible approximation to the eigenvalue λ? Should the closeness of σ to λ be checked?

II. **Direction of starting vector.** How should b be chosen?

III. **Scaling of right hand side.** When the shift σ is very close to an eigenvalue, $\|(A - \sigma I)^{-1}\|$ is large and solving (3.3) may result in overflow. Can $\tau^{(i)}$ be chosen to prevent overflow?

IV. **Convergence Criterion.** When does an iterate $v^{(i)}$ satisfy (2.1)? If the criterion for acceptance is too strict, the iteration may never stop and the danger of too loose a criterion is that poorer approximations than necessary may be accepted.

V. **Orthogonality.** Will the vectors for different eigenvalues computed by (3.3) be numerically orthogonal? If not, what steps must be taken to ensure orthogonality?

As our upcoming discussion will reveal, most of these questions are inter-related and a decision to handle any of the above issues can critically influence other decisions.

We now examine these issues in more detail. Before we do so, it is instructive to look at the first iterate of (3.3) in the eigenvector basis. Suppose that σ is an approximation to the eigenvalue λ_1, and b is the starting vector. Writing b in terms of the eigenvectors, $b = \sum_{i=1}^{n} \xi_i v_i$, we get, in exact arithmetic

$$v^{(1)} = \tau^{(1)}(A - \sigma I)^{-1} b = \tau^{(1)} \left(\frac{\xi_1}{\lambda_1 - \sigma} v_1 + \sum_{i=2}^{n} \frac{\xi_i}{\lambda_i - \sigma} v_i \right)$$

$$\Rightarrow \quad v^{(1)} \;=\; \frac{\xi_1 \tau^{(1)}}{\lambda_1 - \sigma} \left(v_1 + \sum_{i=2}^{n} \frac{\xi_i}{\xi_1} \frac{\lambda_1 - \sigma}{\lambda_i - \sigma} v_i \right). \tag{3.4}$$

I. **Choice of shift.** The above equation shows that for $v^{(1)}$ to be a good approximation to v_1, σ must be close to λ_1. But in such a case, the linear system in (3.3) is ill-conditioned and small changes in σ or A can lead to large changes in the solution $v^{(i+1)}$. Initially, it was feared that roundoff error would destroy these calculations in finite precision arithmetic. However Wilkinson showed that the errors made in computing $v^{(i+1)}$, although large, are almost entirely in the direction of v_1 when λ_1 is isolated. Since we are interested only in computing the direction of v_1 these errors pose no danger, see [17]. Thus to compute the eigenvector of an isolated eigenvalue, the more accurate the shift is, the better is the approximate eigenvector.

   It is common practice nowadays to compute eigenvalues first, and then invoke inverse iteration with very accurate σ. Due to the fundamental limitations of finite precision arithmetic, eigenvalues of symmetric matrixes can, in general, only be computed to a guaranteed accuracy of $O(n\varepsilon\|A\|)$ [14]. Even when a very accurate eigenvalue approximation is available, the following may influence the choice of the shift when more than one eigenvector is desired.

   - **The pairing problem.** In [2], Chandrasekaran gives a surprising example showing how inverse iteration can fail to give small residuals in exact arithmetic if the eigenvalues and eigenvectors are not paired up properly. We reproduce the example in Section 4. To prevent such an occurrence, Chandrasekaran proposes perturbing the eigenvalue approximations so that each shift used for inverse iteration lies to the left of, i.e., is smaller than, the corresponding eigenvalue (see Example 4.1 for more details).
   - **The separation problem.** The solution $v^{(i+1)}$ in (3.3) is very sensitive to small changes in σ when there is more than one eigenvalue near σ. In [20, p.329], Wilkinson notes that

     > 'The extreme sensitivity of the computed eigenvector to very small changes in λ [σ in our notation] may be turned to practical advantage and used to obtain independent eigenvectors corresponding to coincident or pathologically close eigenvalues'.

     Wilkinson proposed that such nearby eigenvalues be 'artificially separated' by a tiny amount in order to compute orthogonal approximations to eigenvectors.

II. **Direction of starting vector.** From (3.4), assuming that $|\lambda_1 - \sigma| \ll |\lambda_i - \sigma|$ for $i \neq 1$, $v^{(1)}$ is a good approximation to v_1 provided that ξ_1 is not "negligible", i.e., the starting vector b must have a non-negligible component in the direction of the desired eigenvector. In [20, pp.315-321], Wilkinson investigates and rejects the choice of e_1 or e_n as a starting vector

(where e_i is the ith column of the $n \times n$ identity matrix). However, it can be shown that e_k is a desirable choice for a starting vector if the kth component of v_1 is above average ($> 1/\sqrt{n}$). In the absence of an efficient procedure to find such a k, Wilkinson proposed choosing PLe as the starting vector, i.e., solving $Uv^{(1)} = e$ in the first iteration, where $T - \sigma I = PLU$ is the LU decomposition obtained by partial pivoting and e is the vector of all 1's [19, 20]. A random starting vector is a popular choice since the probability that it has a negligible component in the desired direction is extremely low, see [13] for a detailed study.

III. **Scaling of right hand side.** Equation (3.4) implies that $\|v^{(1)}\| \geq |\xi_1 \tau^{(1)}|/|\lambda_1 - \sigma|$ where $\tau^{(1)}$ is the scale factor in the first iteration of (3.3). If σ is very close to an eigenvalue, $\|v^{(1)}\|$ can be very large and overflow may occur leading to a breakdown in the eigenvector computation. By choosing $\tau^{(1)}$ appropriately, the right hand side may be scaled down to prevent overflow in certain situations. This approach is taken in EISPACK and LAPACK.

IV. **Convergence Criterion.** In the iteration (3.3), when is $v^{(i+1)}$ an acceptable eigenvector? The residual norm is

$$\frac{\|(A - \sigma I)v^{(i+1)}\|}{\|v^{(i+1)}\|} \;=\; \frac{\|\tau^{(i)} \cdot v^{(i)}\|}{\|v^{(i+1)}\|}. \tag{3.5}$$

The reciprocal of the right hand side, $\|v^{(i+1)}\|/\|\tau^{(i)} \cdot v^{(i)}\|$, is called the norm growth. To guarantee (2.1), $v^{(i+1)}$ is usually accepted when the norm growth is $O(1/n\varepsilon\|A\|)$, see [20, p.324] for details. For the basic iteration of (3.3) this convergence criterion can always be met in a few iterations, provided the starting vector is not pathologically deficient in the desired eigenvector and σ is within $O(n\varepsilon\|A\|)$ of the true eigenvalue. As we have mentioned before, these requirements are easily met.

Since the eigenvalue approximations are generally input to inverse iteration, what should the software do if the input approximations are not accurate, i.e., bad data is input to inverse iteration? We believe that the software should raise some sort of error flag either by testing for the accuracy of the input eigenvalues, or through non-convergence of the iterates.

When an eigenvalue is isolated, a small residual implies orthogonality of the computed vector to other eigenvectors (see (3.6) below). However when eigenvalues are close, goal (2.2) is not automatic. As we now discuss, the methods used to compute numerically orthogonal vectors can impact the choice of the convergence criterion.

V. **Orthogonality.** Standard perturbation theory [14, Section 11-7] says that if $\hat{v}$ is a unit vector, λ is the eigenvalue closest to σ and v is λ's eigenvector then

$$|\sin \angle(v, \hat{v})| \;\leq\; \frac{\|A\hat{v} - \sigma\hat{v}\|}{\text{gap}(\sigma)} \tag{3.6}$$

where $\text{gap}(\sigma) = \min_{\lambda_i \neq \lambda} |\sigma - \lambda_i|$.

In particular, the above implies that the simple iteration scheme of (3.3) cannot guarantee orthogonality of the computed "eigenvectors" when eigenvalues are close. To achieve numerical orthogonality, current implementations modify (3.3) by explicitly orthogonalizing each iterate against previously computed eigenvectors of nearby eigenvalues.

However, orthogonalization can fail to give the desired answer if the vectors are nearly linearly dependent prior to the orthogonalization. Two difficulties arise in such a situation :

```
    INVERSE ITERATION(A,λ̂)
/* assume that v̂_1, v̂_2, ..., v̂_{j-1} have been computed, and that λ̂_i, λ̂_{i+1}, ..., λ̂_j form a cluster */
    Choose a starting vector b_j;
    Orthogonalize b_j against v̂_i, v̂_{i+1}, ..., v̂_{j-1};
    l = 0; v^(0) = b_j;
    do
        l = l + 1;
        Solve (A − λ̂_j I)v^(l) = τ^(l−1) v^(l−1);
        Orthogonalize v^(l) against v̂_i, v̂_{i+1}, ..., v̂_{j-1};
    while(‖v^(l)‖/‖τ^(l−1) v^(l−1)‖ is not "big" enough)
    v̂_j = v^(l)/‖v^(l)‖;
```

Figure 1: A basic implementation of Inverse Iteration to compute the jth eigenvector

- The orthogonalized vectors may not provide an orthogonal basis of the desired subspace.
- Orthogonalization may lead to cancellation and a decrease in norm of the iterate. Thus a simple convergence criterion (as suggested above in issue IV) may not be reached.

As we shall show in Section 4.1, the above situation can occur surprisingly often due to inaccuracies in the computed eigenvalues. In particular, approximations to small eigenvalues are often incorrect in *all* their digits (eigenvalues found by the QR algorithm and the Divide and Conquer method can have an error as large as $O(\varepsilon\|T\|)$). In response to such problems, Chandrasekaran proposed a new version of inverse iteration in [2] that is considerably different from the EISPACK and LAPACK implementations. The differences include an alternate convergence criterion. The drawback of this new version is the potential increase in the amount of computation required.

4 Existing Implementations

Figure 1 gives the pseudocode for a basic implementation of inverse iteration to compute v_j, the jth eigenvector, assuming that approximations to $v_1, v_2, \ldots, v_{j-1}$ have already been computed. Note that in this pseudocode, both the starting vectors and iterates are orthogonalized against previously computed eigenvectors. Surprisingly, as the following example shows, this implementation can fail to give small residual norms *even in exact arithmetic* by incorrectly pairing up the eigenvalues and eigenvectors.

Example 4.1 [The Pairing Error.] (Chandrasekaran [2]) Let λ_1 be an arbitrary real number, and

$$\lambda_2 = \lambda_1 + \varepsilon, \quad \lambda_{i+1} - \lambda_i = \lambda_i - \lambda_1, \quad \lambda_n \geq 1, \quad i = 2, \ldots, n-1$$

where ε is the machine precision. Explicitly, $\lambda_{i+1} = \lambda_1 + 2^{i-1}\varepsilon$. Suppose that

$$\hat{\lambda}_i > \lambda_i, \quad i = 1, \ldots, n \qquad \text{and most importantly} \qquad \hat{\lambda}_1 - \lambda_1 > \lambda_2 - \hat{\lambda}_1.$$

Figure 2: Eigenvalue Distribution in example

Figure 2 illustrates the situation.

Suppose that we take a conservative approach and orthogonalize each starting vector b_j against *all* previously computed eigenvectors $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_{j-1}$ (see Figure 1). It can be shown that if $b_j^T v_{j+1} \neq 0$, then in exact arithmetic the computed eigenvectors are

$$\hat{v}_j = v_{j+1}, \quad j = 1, \ldots, n-1$$

and, because $\hat{v}_n$ must be orthogonal to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_{n-1}$, the last computed eigenvector is

$$\hat{v}_n = v_1.$$

Since the eigenvalues grow exponentially, the residual norm $\|(A - \hat{\lambda}_n I)\hat{v}_n\|$ is large. □

Thus a simple inverse iteration code based on orthogonalization may appear to fail even in exact arithmetic. To cure this problem, Chandrasekaran proposed that $\hat{\lambda}_i - O(n\varepsilon\|A\|)$ be used as the shifts for inverse iteration so that each shift is guaranteed to lie to the left of its true eigenvalue [2]. As we shall see later, neither EISPACK nor LAPACK perform this 'artificial' perturbation.

The discerning reader will realize that the above problem is not the failure of the basic inverse iteration process. Inverse Iteration does the work it is designed for, since the iterates do converge to the closest eigenvector that is orthogonal to the previously computed eigenvectors. The error is in the incorrect pairing of the eigenvalues and eigenvectors. Once this elusive error is seen, it may be argued that such a simple implementation is sloppy. An easy cure, that would correctly pair up the eigenvalues and eigenvectors, would be to associate each computed eigenvector with its Rayleigh Quotient, which is available at a modest cost. In particular, the residual norm of each eigenvector can be measured with respect to its Rayleigh Quotient, i.e., as

$$\|A\hat{v}_i - (\hat{v}_i^T A\hat{v}_i)\hat{v}_i\|,$$

where we have replaced the input eigenvalue, $\hat{\lambda}_i$, by $\hat{v}_i^T A\hat{v}_i$. Unfortunately, because of the premium on speed, most current software does not perform such *posteriori* corrections nor does it verify the correctness of its output. Thus, errors can go undetected since the task of proving correctness of numerical software is often compromised by testing it on a finite sample of a multi-dimensional infinite space of inputs.

We now look in detail at two existing implementations of inverse iteration and see how they address the issues discussed in the previous section. EISPACK [18] and LAPACK [1] are linear algebra software libraries that contain routines to solve various eigenvalue problems. EISPACK's implementation of inverse iteration is named INVIT while LAPACK's inverse iteration subroutine is called xSTEIN[1] (STEIN is an acronym for Symmetric Tridiagonal's Eigenvectors through Inverse

[1]The prefix 'x' stands for the data type: real single(S) or real double(D), or complex single(C) or complex double(Z).

Iteration). xSTEIN was developed to be more accurate than TINVIT as the latter was found to deliver less than satisfactory results in several test cases. In order to achieve accuracy comparable to that of the Divide and Conquer and QR/QL methods, the search for a better implementation of inverse iteration led to xSTEIN [13]. However, as we will see in Section 4.1, xSTEIN also suffers from some of the same problems as TINVIT in addition to introducing a new serious error.

Both EISPACK and LAPACK solve the dense symmetric eigenproblem by reducing the dense matrix to tridiagonal form by Householder transformations [11], and then finding the eigenvalues and eigenvectors of the tridiagonal matrix. Both TINVIT and xSTEIN operate on a symmetric tridiagonal matrix. In the following, we will further assume that the tridiagonal is unreduced, i.e., all its off-diagonal elements are nonzero.

4.1 EISPACK and LAPACK Implementations

Figure 3 gives the pseudocode for TINVIT [18, 16] while Figure 4 outlines the pseudocode for xSTEIN as it appears in LAPACK release 2.0. The latter code has changed little since it was first released in 1992. It is not necessary for the reader to absorb all details of the implementations given in Figures 3 and 4 to follow the ensuing discussion. We provide the pseudocodes as references in case the reader needs to look in detail at particular aspects of the implementations.

In each iteration, TINVIT and xSTEIN solve the scaled linear system $(T-\hat{\lambda}I)y = \tau b$ by Gaussian Elimination with partial pivoting. If eigenvalues agree in more than three digits relative to the norm, the iterates are orthogonalized against previously computed eigenvectors by the modified Gram-Schmidt method. Note that in both these routines the starting vector is not made orthogonal to previously computed eigenvectors, as is done in Figure 1. Both TINVIT and xSTEIN flag an error if the convergence criterion is not satisfied within five iterations. To achieve greater accuracy, xSTEIN does two extra iterations after the stopping criterion is satisfied. We now compare and contrast how these implementations handle the various issues discussed in Section 3.

I. **Choice of shift.** Even though in exact arithmetic all the eigenvalues of an unreduced tridiagonal matrix are distinct, some of the computed eigenvalues may be identical to working accuracy. In [20, p.329], Wilkinson recommends that pathologically close eigenvalues be perturbed by a small amount in order to get an orthogonal basis of the desired subspace. Following this, TINVIT replaces equal approximations $\hat{\lambda}_j = \hat{\lambda}_{j+1} = \cdots = \hat{\lambda}_{j+k}$ by

$$\hat{\lambda}_j < \hat{\lambda}_j + \varepsilon\|T\|_R < \cdots < \hat{\lambda}_j + k\varepsilon\|T\|_R,$$

where $\|T\|_R = \max_i |T_{ii}| + |T_{i,i+1}| \leq \|T\|_1$.

We now give an example where this perturbation is too big. As a result, the shifts used to compute the eigenvectors are quite different from the true eigenvalues and prevent the convergence criterion from being attained.

Example 4.2 [Excessive Perturbation.] Using LAPACK's test matrix generator [4], we generated a 200×200 tridiagonal matrix such that

$$\hat{\lambda}_1 \approx \cdots \approx \hat{\lambda}_{100} \approx -\varepsilon, \qquad \hat{\lambda}_{101} \approx \cdots \approx \hat{\lambda}_{199} \approx \varepsilon, \qquad \hat{\lambda}_n = 1$$

```
TINVIT(T,λ̂)
/*  TINVIT computes all the eigenvectors of T given the computed eigenvalues
    λ̂_1, λ̂_2, ..., λ̂_n arranged in non-decreasing order */
for j = 1, n
    σ_j = λ̂_j;
    if j > 1 and λ̂_j ≤ σ_{j-1} then
        σ_j = σ_{j-1} + ε‖T‖_R; /* Perturb identical eigenvalues */
    end if
    Factor T − σ_j I = PLU; /* Gaussian Elimination with partial pivoting */
    if U_nn == 0 then U_nn = ε‖T‖;
    τ = √n ε‖T‖_R; /* Compute scale factor */
    Solve Uy = τe; /* Solve. Here, e is the vector of all 1's */
    for all k < j such that |σ_j − σ_k| ≤ 10^{-3}‖T‖_R
        y = y − (y^T v̂_k)v̂_k; /* Apply Modified Gram-Schmidt */
    end for
    b = y; iter = 1;
    while(‖y‖_1 < 1 and iter ≤ 5) do
        τ = nε‖T‖_R/‖b‖_1; /* Compute scale factor */
        Solve PLUy = τb; /* Solve with scaled right hand side */
        for all k < j such that |σ_j − σ_k| ≤ 10^{-3}‖T‖_R do
            y = y − (y^T v̂_k)v̂_k; /* Apply Modified Gram-Schmidt */
        end for
        b = y; iter = iter + 1;
    end while
    if ‖y‖_1 < 1 then
        v̂_j = 0; ierr = −j; /* set error flag */
        print "jth eigenvector failed to converge";
    else
        v̂_j = y/‖y‖_2;
    end if
end for
```

Figure 3: TINVIT — EISPACK's implementation of Inverse Iteration

```
xStein(T,λ̂)
/*  xStein computes all the eigenvectors of T given the computed eigenvalues
    λ̂_j, 1 ≤ j ≤ n, arranged in non-decreasing order */
for j = 1, n
    σ_j = λ̂_j;
    if j > 1 and λ̂_j − σ_{j−1} ≤ 10ε|λ̂_j| then
        σ_j = σ_{j−1} + 10ε|λ̂_j|; /* Perturb nearby eigenvalues */
    end if
    Factor T − σ_j I = PLU; /* Gaussian Elimination with partial pivoting */
    Initialize b = random vector;
    iter = 0; extra = 0; converged = false;
    do
        τ = n‖T‖_1 max(ε, |U_nn|)/‖b‖_1; /* Compute scale factor */
        Solve PLUy = τb; /* Solve with scaled right hand side */
        for all k < j such that |σ_j − σ_k| ≤ 10^{−3}‖T‖_1 do
            y = y − (y^T v̂_k)v̂_k; /* Apply Modified Gram-Schmidt */
        end for
        b = y; iter = iter + 1;
        if converged == true then extra = extra + 1; end if
        if ‖y‖_∞ ≥ sqrt(1/(10n)) then converged = true; end if
    while((converged == false or extra < 2) and iter ≤ 5)
    v̂_j = y/‖y‖_2;
    if iter > 5 and extra < 2 then
        print "jth eigenvector failed to converge";
    end if
end for
```

Figure 4: xStein — LAPACK's implementation of Inverse Iteration

where $\varepsilon \approx 1.2 \times 10^{-7}$ (this run was in single precision). $\|T\|_R = O(1)$, and the shift used by TINVIT to compute $\hat{v}_{199}$ is

$$\sigma = -\varepsilon + 198\varepsilon\|T\|_R \approx 2.3 \times 10^{-5}.$$

The error $|\sigma - \lambda_{199}|$ can be as large as

$$|\sigma - \hat{\lambda}_{199}| + |\hat{\lambda}_{199} - \lambda_{199}| \approx 2.3 \times 10^{-5} + n\varepsilon\|T\|_R \approx 4.6 \times 10^{-5}.$$

The norm growth when solving $(T-\sigma I)y = \tau b$ is inversely proportional to this error (see (4.8) and (4.9) below). Consequently, the norm growth narrowly misses the convergence criterion of $\|y\| \geq 1$ in a numerical run. TINVIT flags this as an error and returns $ierr = -199$. □

As shown above, perturbing coincident eigenvalues by $\varepsilon\|T\|$ can substantially degrade their accuracy. However, not perturbing them is also not acceptable in TINVIT as coincident shifts lead to identical $L\,U$ factors and starting vectors, which ultimately result in eigenvectors that are parallel prior to reorthogonalization.

In xSTEIN, coincident eigenvalues are also perturbed. However, the perturbations made are small *relative* to each eigenvalue. In particular, equal approximations $\hat{\lambda}_j = \hat{\lambda}_{j+1} = \cdots = \hat{\lambda}_{j+k}$ are replaced by

$$\hat{\lambda}_j < \hat{\lambda}_{j+1} + \delta\hat{\lambda}_{j+1} < \cdots < \hat{\lambda}_{j+k} + \delta\hat{\lambda}_{j+k},$$

where $\delta\hat{\lambda}_i = \sum_{l=j+1}^{i-1} \delta\hat{\lambda}_l + 10\varepsilon|\hat{\lambda}_i|$. This choice does not perturb small eigenvalues drastically, and appears to be better than TINVIT's. On the other hand, this perturbation is too small in some cases to serve its purpose of finding a linearly independent basis of the desired subspace (see Example 4.7, and Wilkinson's quote given on page 4). Thus it is easier to say "tweak close eigenvalues" than to find a satisfactory formula for it.

II. **Direction of starting vector.** TINVIT chooses the starting vector to be PLe where $T-\sigma I = PLU$ is obtained by partial pivoting, σ being the input eigenvalue approximation and e the vector of all 1's. Note that this choice of starting vector reduces the first iteration to simply solving $Uy = \tau e$. On the other hand, xSTEIN chooses a random starting vector, each of whose elements comes from a uniform $(-1, 1)$ distribution. Neither choice of starting vectors is likely to be pathologically deficient in the desired eigenvector. The random starting vectors are designed to be superior to TINVIT's choice [13].

III. **Scaling of right hand side.** Both TINVIT and xSTEIN solve the linear system

$$(T - \sigma I)y \;=\; \tau b \qquad \text{as} \qquad PLUy \;=\; \tau b \tag{4.7}$$

at each iteration. Suppose that σ is an approximation to λ_1. By analysis similar to (3.4), we see that if b is a good starting vector then

$$\|y\| = O\left(\frac{\tau\|b\|}{|\lambda_1 - \sigma|}\right). \tag{4.8}$$

Clearly, when σ is close to λ_1, $\|y\|$ can be large. TINVIT attempts to prevent overflow when computing y by choosing the scale factor as

$$\tau \;=\; \frac{n\varepsilon\|T\|_R}{\|b\|_1}. \tag{4.9}$$

However, if $|\lambda_1 - \sigma| \ll n\varepsilon\|T\|_R$ then (4.8) implies that $\|y\|$ can still be very large. TINVIT handles this problem only partially — it prevents a division by zero in (4.7) by replacing a zero value of the last pivot, U_{nn}, with $\varepsilon\|T\|$. However, as the following example shows, overflow can still occur.

Example 4.3 [Perturbing zero values is not enough.] Let

$$T = \begin{bmatrix} -\eta & 10 & 0 \\ 10 & 0 & 10 \\ 0 & 10 & \eta(1+\varepsilon) \end{bmatrix} \tag{4.10}$$

where η is the underflow threshold of the machine ($\eta \approx 10^{-308}$ in IEEE double precision arithmetic). T is nearly singular and with the shift $\sigma = 0$,

$$\text{partial pivoting in } \; T - \sigma I = PLU \quad \Rightarrow \quad U_{nn} = \eta\varepsilon.$$

In the first iteration, $Uy = \tau e$ and $\tau = \sqrt{n}\varepsilon\|T\|_R$, which leads to

$$y(n) \;=\; \frac{\tau}{U_{nn}} \;=\; \frac{\sqrt{n}\varepsilon\|T\|_R}{\eta\varepsilon} \;\approx\; \frac{10\sqrt{n}}{\eta} \quad \Rightarrow \quad \text{overflow!}$$

Note that to exhibit the above failure, we required gradual underflow in IEEE arithmetic (we required $U_{nn} = \eta\varepsilon$ to not underflow to zero). However failure can be observed even on non-IEEE machines — a similar error, where there is no gradual underflow, occurs on $(1/\varepsilon)T$ where T is as in (4.10). □

In contrast, xSTEIN chooses its scale factor to be

$$\tau \;=\; \frac{n\|T\|_1 \max(\varepsilon, |U_{nn}|)}{\|b\|_1}. \tag{4.11}$$

The only significant difference from the choice in TINVIT is the term $\max(\varepsilon, |U_{nn}|)$ instead of ε [1, 12]. However this difference introduces a serious error not present in TINVIT, which we now explain. Suppose σ approximates λ_1 in (4.7). When $\sigma = \lambda_1$, it can be proved that U_{nn} must be zero in exact arithmetic. We now examine the values U_{nn} takes when $\sigma \neq \lambda_1$ (we assume that $|\sigma - \lambda_1| \ll |\sigma - \lambda_i|, i \neq 1$). Since $T - \sigma I = PLU$,

$$\begin{aligned} U^{-1}L^{-1} &= (T - \sigma I)^{-1}P \\ \Rightarrow \; e_n^T U^{-1}L^{-1}e_n &= e_n^T(T - \sigma I)^{-1}Pe_n. \end{aligned}$$

Since L is unit lower triangular, $L^{-1}e_n = e_n$. Letting $Pe_n = e_k$ and $T = V\Lambda V^T$, we get

$$\begin{aligned} \frac{1}{U_{nn}} &= e_n^T V(\Lambda - \sigma I)^{-1} V^T e_k \\ \Rightarrow \quad \frac{1}{U_{nn}} &= \frac{v_1(n)\cdot v_1(k)}{\lambda_1 - \sigma} + \sum_{i=2}^{n} \frac{v_i(n)\cdot v_i(k)}{\lambda_i - \sigma}, \end{aligned} \tag{4.12}$$

where $v_i(k)$ denotes the kth component of v_i. By examining the above equation closely, we see that $|U_{nn}|$ is "small" only if $|\lambda_1 - \sigma| \ll |v_1(n)\cdot v_1(k)|$. When the latter condition is not satisfied, $|U_{nn}|$ can be much larger than ε and its value does not reveal the nearness of σ to an eigenvalue of T. In such a case, (4.11) implies that τ is also larger and as the next two examples, this can lead to a myriad of problems.

Example 4.4 [A code may fail but should never lie.] Consider

$$T = \begin{bmatrix} 1 & \sqrt{\varepsilon} & 0 \\ \sqrt{\varepsilon} & 7\varepsilon/4 & \varepsilon/4 \\ 0 & \varepsilon/4 & 3\varepsilon/4 \end{bmatrix} \tag{4.13}$$

where ε is the machine precision ($\varepsilon \approx 2.2 \times 10^{-16}$ in IEEE double precision arithmetic). T has eigenvalues near $\varepsilon/2, \varepsilon, 1+\varepsilon$. Suppose an eigenvalue approximation *is incorrectly input as* 2. Then by (4.11) and (4.12),

$$\sigma = 2 \quad \Rightarrow \quad |U_{nn}| = O(1) \quad \Rightarrow \quad \tau = O(1)$$

if $\|b\| = 1$. This means that the right hand side b is not scaled down in (4.7). Thus a large norm growth, which would imply a small residual norm, is not ensured when the stopping criterion is satisfied, see Figure 4, (4.8) and (4.15) below. As a result, any arbitrary vector will be accepted as an approximate eigenvector — in a numerical run, the vector $[-0.6446 \;\; 0.6373 \;\; 0.4223]^T$ was output by xSTEIN even though it is nowhere close to any eigenvector of T. □

The above is clearly a *bug* in xSTEIN since the convergence criterion does not guarantee a small residual norm. This example represents one of the more dangerous errors of numerical software — the software outputs an incorrect answer but does not flag any error at all. The reader should recall that the goal of inverse iteration is to produce vectors that satisfy (2.1) and (2.2). When the input eigenvalue approximations do not permit such high accuracy, the correct response would be to signal a failure to converge to the desired vectors (the author would like to emphasize that inability to handle incorrect input data can have disastrous consequences [2]). Note that on the above example, TINVIT correctly flags an error indicating that the computation did not "converge".

Even if σ is a very good approximation to λ_1, (4.12) indicates that U_{nn} may not be small if $v_1(n)$ is tiny. It turns out that a component of an eigenvector of a tridiagonal matrix can

[2] In the summer of 1996, a core dump on the main computer aboard the Ariane 5 rocket was interpreted as flight data, causing a violent trajectory correction that led to the disintegration of the rocket

frequently be very small, see [20, pp.317-321]. In such a case, xSTEIN can choose an unnecessary large scale factor which can occasionally lead to overflow, as shown by the following example.

Example 4.5 [Undeserved overflow.] Consider the matrix given above in (4.13). The eigenvector corresponding to the eigenvalue $\lambda_1 = 1 + \varepsilon + O(\varepsilon^2)$ is

$$v_1 = \begin{bmatrix} 1 - \varepsilon/2 + O(\varepsilon^3) \\ \sqrt{\varepsilon} + O(\varepsilon^{3/2}) \\ \varepsilon^{3/2}/4 + O(\varepsilon^{5/2}) \end{bmatrix}.$$

If $\sigma = 1$, then $|v_1(n)|/|\lambda_1 - \sigma| < \sqrt{\varepsilon}$ and the $\sum_{i=2}^{n}$ term in (4.12) is dominant implying that $|U_{nn}| = O(\|T\|)$. Combined with (4.11), this gives $\tau = O(\|T\|^2)$. Thus if $\|T\| > 1$, the right hand side is scaled *up* instead of the usual practice of being scaled *down*. As a consequence, xSTEIN overflows on the scaled matrix $\sqrt{\Omega}\,T$ where Ω is the overflow threshold of the computer ($\Omega = 2^{1023} \approx 10^{308}$ in IEEE double precision arithmetic). □

Note that the above matrix $\sqrt{\Omega}\,T$ does not deserve overflow. A similar overflow occurrence (in IEEE double precision arithmetic) on an 8×8 matrix, with a largest element of magnitude $2^{484} \approx 10^{145}$, was reported to us by Jeremy DuCroz [7].

The problems reported in the above two examples can be cured by reverting back to the choice of scale factor in EISPACK's TINVIT.

IV. **Convergence Criterion.** Both TINVIT and xSTEIN judge the quality of an approximate eigenvector by the norm growth obtained in solving the linear system (4.7). For example, TINVIT accepts y as an eigenvector if

$$\|y\|_1 \geq 1. \tag{4.14}$$

Note that (4.7) and (4.9) imply that the residual norm is

$$\frac{\|(T - \sigma I)y\|_1}{\|y\|_1} \;=\; \frac{\tau\|b\|_1}{\|y\|_1} \;=\; \frac{n\varepsilon\|T\|_R}{\|y\|_1}.$$

Thus TINVIT's convergence criterion (4.14) ensures a small residual norm, see (2.2). From (4.8) and (4.9), it is clear that for $\|y\|$ to be bigger than 1, the shift σ must be within $O(n\varepsilon\|T\|)$ of a true eigenvalue. If the input approximation is less accurate, the iterates do not "converge" and TINVIT flags an error.

xSTEIN has a similar stopping criterion; it accepts y if

$$\|y\|_\infty \geq \sqrt{\frac{1}{10n}}. \tag{4.15}$$

However, as discussed earlier, the choice of scale factor in xSTEIN is unfortunate and the above convergence criterion does not guarantee a small residual norm. Consequently, xSTEIN can output poor approximations to eigenvectors without signaling an error, see Example 4.4 for such an instance.

V. **Orthogonality.** TINVIT and xSTEIN use the modified Gram-Schmidt (MGS) procedure to orthogonalize iterates corresponding to eigenvalues whose separation is less than $10^{-3}\|T\|$. In order for MGS to produce numerically orthogonal vectors, it is crucial that the vectors to be orthogonalized be numerically linearly independent. However, as the following example shows, inaccuracies in the small eigenvalues can lead to vectors that are almost parallel prior to MGS.

Example 4.6 [Parallel Iterates.] Consider the matrix of (4.13). T has the eigenvalues

$$\lambda_1 = \varepsilon/2 + O(\varepsilon^2), \qquad \lambda_2 = \varepsilon + O(\varepsilon^2), \qquad \lambda_3 = 1 + \varepsilon + O(\varepsilon^2).$$

The eigenvalues of T as computed by MATLAB's **eig**[3] function (on an IBM RS/6000-590 processor[4]) are

$$\hat{\lambda}_1 = \varepsilon, \qquad \hat{\lambda}_2 = \varepsilon, \qquad \hat{\lambda}_3 = 1 + \varepsilon.$$

Note that $\hat{\lambda}_1 = \hat{\lambda}_2$ and $\hat{\lambda}_1$ is closer to λ_2 than to λ_1. We perturb $\hat{\lambda}_2$ to $\varepsilon(1+\varepsilon)$ and input these approximations to TINVIT to demonstrate failure (since otherwise TINVIT would perturb $\hat{\lambda}_2$ by $\varepsilon\|T\|$, see Example 4.2).

The first eigenvector is computed by TINVIT as

$$y_1 = (T - \hat{\lambda}_1 I)^{-1} b_1.$$

In exact arithmetic (taking $b_1 = \sum_i \xi_i v_i$),

$$\begin{aligned} y_1 &= \frac{\xi_2}{\lambda_2 - \hat{\lambda}_1}\left(v_2 + \frac{\xi_1}{\xi_2}\frac{\lambda_2 - \hat{\lambda}_1}{\lambda_1 - \hat{\lambda}_1}v_1 + \frac{\xi_3}{\xi_2}\frac{\lambda_2 - \hat{\lambda}_1}{\lambda_3 - \hat{\lambda}_1}v_3\right) \\ &= \frac{1}{O(\varepsilon^2)}\left(v_2 + O(\varepsilon)v_1 + O(\varepsilon^2)v_3\right) \end{aligned}$$

where we have assumed that b_1 is a good starting vector, i.e., $\xi_2 = b_1^T v_2$ is $O(1)$. Due to the inevitable roundoff errors in finite precision, the best we can hope to compute is

$$\hat{y}_1 = \frac{1}{O(\varepsilon^2)}\left(v_2 + O(\varepsilon)v_1 + O(\varepsilon)v_3\right).$$

This vector is then normalized to remove the $1/O(\varepsilon^2)$ factor and give the first approximate eigenvector $\hat{v}_1$. However, note that $\hat{v}_1$ is actually a very good approximation to the *second* eigenvector v_2. This should not surprise us as $\hat{\lambda}_1$ is much closer to λ_2 than to λ_1.

The second eigenvector is now computed as

$$y_2 = (T - \hat{\lambda}_2 I)^{-1} b_2.$$

[3] MATLAB's **eig** function computes eigenvalues by the QR algorithm.

[4] we suspect that **eig** returns this set of eigenvalues, where $\hat{\lambda}_1 = \hat{\lambda}_2$, on all processors that do not accumulate inner products, such as the IBM RS/6000-590, SUN Ultra and Mac Power PC 604.

Since $\hat{\lambda}_2 \approx \hat{\lambda}_1$, the computed value of y_2 is also nearly parallel to v_2 (assuming that b_2 has a non-negligible component in the direction of v_2), i.e.,

$$\hat{y}_2 = \frac{1}{O(\varepsilon^2)} \left(v_2 + O(\varepsilon) v_1 + O(\varepsilon) v_3 \right).$$

As $\hat{\lambda}_1$ and $\hat{\lambda}_2$ are nearly coincident, TINVIT orthogonalizes $\hat{y}_2$ against $\hat{v}_1$ by the MGS process in an attempt to reveal the second eigenvector :

$$z = \hat{y}_2 - (\hat{y}_2^T \hat{v}_1) \hat{v}_1 \tag{4.16}$$

However, since $\hat{y}_2$ and $\hat{v}_1$ are nearly parallel, the severe cancellation in the above step leaves a vector z which, essentially, is random noise :

$$\begin{aligned} z &= \frac{1}{O(\varepsilon^2)} \left(O(\varepsilon) v_1 + O(\varepsilon) v_2 + O(\varepsilon) v_3 \right), \\ \hat{v}_2 &= z / \|z\|. \end{aligned}$$

Clearly z is not orthogonal to $\hat{v}_1$. But TINVIT accepts z as having "converged" since $\|z\|$ is big enough to satisfy the convergence criterion (4.14) even after the severe cancellation in (4.16). The above observations are confirmed by a numerical run in double precision arithmetic where the first two eigenvectors output by TINVIT had a dot product of 0.0452. □

Unlike TINVIT, xSTEIN computes each eigenvector from a different random starting vector. The hope is to get greater linear independence of the iterates before the MGS step [13]. However, as we now show, the TINVIT error as reported above persists due to inaccuracies in the small eigenvalues.

Example 4.7 [Linear Dependence Persists.] Consider again the matrix T given in (4.13). The eigenvalues input to xSTEIN are computed by the **eig** function in MATLAB as $\hat{\lambda}_1 = \hat{\lambda}_2 = \varepsilon$ and $\hat{\lambda}_3 = 1 + \varepsilon$ (see the footnote at the bottom of page 15). As in TINVIT, the first computed eigenvector $\hat{v}_1$ is almost parallel to v_2. The iterations to compute the second eigenvector are summarized in Table 1. Note that we have listed the last few digits of the iterates prior to MGS to indicate the small changes in the vectors.

From Table 1 we see that at every iteration, the vector y is observed to become parallel to $\hat{v}_1$ just before MGS. This behavior is very similar to that of TINVIT (see Example 4.6). xSTEIN does two more iterations than TINVIT and alleviates the problem slightly, but a dot product of 1.9×10^{-6} between computed eigenvectors is far from satisfactory (this run was in double precision). □

xSTEIN avoids the overflow problems of TINVIT exhibited in Example 4.3. It checks to see if overflow would occur, and if so, perturbs tiny entries on the diagonal of U [1, 12]. This check is in the inner loop when solving $Uy = x$ where $x = \tau L^{-1} P^{-1} b$. Coupled with the extra iterations done after convergence, this results in xSTEIN being slower than TINVIT. On an assorted collection of test matrices of sizes 50 to 1000, we observed xSTEIN to be 2-3 times slower than TINVIT. However xSTEIN was more accurate than TINVIT in general.

Step	1		2		3	
	Before MGS	After MGS	Before MGS	After MGS	Before MGS	After MGS
Iterate(y)	$-1.05 \cdot 10^{-8}$ $.707\ldots5471$ $.707\ldots5477$	$-1.04 \cdot 10^{-8}$ $.697$ $-.716$	$-1.05 \cdot 10^{-8}$ $.707\ldots5351$ $.707\ldots5599$	$-1.05 \cdot 10^{-8}$ $.7069$ $-.7073$	$-1.05 \cdot 10^{-8}$ $.707\ldots9587$ $.707\ldots1363$	$-1.05 \cdot 10^{-8}$ $.707108$ $-.707105$
$y^T\hat{v}_1$	1.000	.0014	1.000	$3.0 \cdot 10^{-4}$	1.000	$1.9 \cdot 10^{-6}$
$y^T\hat{v}_3$	$3.9 \cdot 10^{-24}$	$4.1 \cdot 10^{-11}$	$5.8 \cdot 10^{-25}$	$2.9 \cdot 10^{-12}$	$2.2 \cdot 10^{-24}$	$1.1 \cdot 10^{-13}$

Table 1: Summary of xSTEIN's iterations to compute the second eigenvector of T.

5 Conclusions and Future Work

In this paper, we have demonstrated how existing EISPACK and LAPACK implementations of inverse iteration can fail. The modes of failure are revealed by a systematic examination of decisions made in the software and their impact on the dual goals of small residual norms and orthogonality. Some of the "worst" errors are easily rectified. We now indicate the current state of knowledge about the various aspects of inverse iteration discussed in this paper, and mention alternate approaches to the unresolved issues.

I. **Choice of shift.** Of the various issues discussed in this paper, the choice of starting vector and convergence criterion have been extensively studied [20, 16, 17, 13]. Surprisingly, the choice of shift for inverse iteration seems to have drawn little attention. We feel that the shift is probably the most important variable in inverse iteration. Examples 4.6 and 4.7 highlight the importance of shifts that are *as accurate as possible.*

II. & III. **Direction of starting vector and scaling of right hand side.** *This particular problem has recently benn solved.* It is now possible to deterministically find a starting vector that is guaranteed to have an above average component in the direction of the desired eigenvector v. Knowing the position of a large component of v also enables us to avoid the possibility of overflow in the eigenvector computation. See [5, 15, 8, 10, 9] for more details.

IV. & V. **Convergence criterion and Orthogonality.** It is easy to find a criterion that guarantees small residual norms, see goal (2.1). However, as we saw in earlier sections, the goal of orthogonality (2.2) can lead to a myriad of problems. Most of the "difficult" errors in the EISPACK and LAPACK implementations arise due to the explicit orthogonalization of iterates when eigenvalues are close. In [2], Chandrasekaran gives an explanation of these failures, and proposes a more robust version of inverse iteration. However, this new version is more involved and potentially requires much more orthogonalization than existing implementations. There is no current plan to widely distribute software based on this new version of inverse iteration [3]. Recently, there has been much work on another approach for computing numerically orthogonal approximations to eigenvectors. However this work is still ongoing and beyond the scope of this paper. The interested reader is requested to see [5, 15] and await [6].

Acknowledgements. I would like to thank Professors B.N.Parlett, J.W.Demmel, Axel Ruhe and an anonymous referee for a careful reading of the manuscript and for many useful suggestions.

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 (second edition)*. SIAM, Philadelphia, 1995. 324 pages.

[2] S. Chandrasekaran. *When is a Linear System Ill-Conditioned?* PhD thesis, Departement of Computer Science, Yale University, New Haven, CT, 1994.

[3] S. Chandrasekaran, 1996. private communication.

[4] J. Demmel and A. McKenney. A test matrix generation suite. Computer science dept. technical report, Courant Institute, New York, NY, July 1989. (LAPACK Working Note #9).

[5] I. S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, Computer Science Division, University of California, Berkeley, California, May 1997. Available as UC Berkeley Technical Report No. UCB//CSD-97-971.

[6] I.S. Dhillon and B.N. Parlett. Computing the eigenvectors of a symmetric tridiagonal matrix. in preparation.

[7] J. DuCroz, December 1994. private communication.

[8] K. V. Fernando. On computing an eigenvector of a tridiagonal matrix. Part I: Basic results. *SIAM J. Matrix Anal. Appl.*, 18(4):1013–1034, October 1997.

[9] S. K. Godunov, A. G. Antonov, O. P. Kiriljuk, and V. I. Kostin. *Guaranted Accuracy in Numerical Linear Algebra*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993. A revised translation of a Russian text first published in 1988 in Novosibirsk.

[10] S. K. Godunov, V. I. Kostin, and A. D. Mitchenko. Computation of an eigenvector of symmetric tridiagonal matrices. *Siberian Math. J.*, 26:71–85, 1985.

[11] Alston S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comput. Mach.*, 5:339–342, 1958.

[12] I. C. F. Ipsen. Computing an eigenvector with inverse iteration. *SIAM Review*, 39(2):254–291, 1997.

[13] E. Jessup and I. Ipsen. Improving the accuracy of inverse iteration. *SIAM J. Sci. Stat. Comput.*, 13(2):550–572, 1992.

[14] B. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, second edition, 1997.

[15] B. N. Parlett and I. S. Dhillon. Fernando's solution to Wilkinson's problem: An application of double factorization. *Lin. Alg. Appl.*, 267:247–279, December 1997.

[16] G. Peters and J.H. Wilkinson. *The calculation of specified eigenvectors by inverse iteration, contribution II/18*, volume II of *Handbook of Automatic Computation*, pages 418–439. Springer-Verlag, New York, Heidelberg, Berlin, 1971.

[17] G. Peters and J.H. Wilkinson. Inverse iteration, ill-conditioned equations and Newton's method. *SIAM Review*, 21:339–360, 1979.

[18] 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.

[19] J. H. Wilkinson. The calculation of the eigenvectors of codiagonal matrices. *Computer J.*, 1:90–96, 1958.

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