Available online at www.sciencedirect.com
ScienceDirect
Linear Algebra and its Applications 422 (2007) 77–99
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

Computation of exact inertia and inclusions of eigenvalues (singular values) of tridiagonal (bidiagonal) matrices

K.V. Fernando *

Division of Structural Biology, The Wellcome Trust Centre for Human Genetics, University of Oxford, Roosevelt Drive, Headington, Oxford OX3 7BN, UK

Received 27 December 2005; accepted 10 September 2006
Available online 30 October 2006
Submitted by J.L. Barlow

Dedicated to William Kahan

Abstract

This report may be considered as a non-trivial extension of an unpublished report by William Kahan (*Accurate Eigenvalues of a symmetric tri-diagonal matrix*, Technical Report CS 41, Computer Science Department, Stanford University, 1966). His interplay between matrix theory and computer arithmetic led to the development of algorithms for computing accurate eigenvalues and singular values. His report is generally considered as the precursor for the development of IEEE standard 754 for binary arithmetic. This standard has been universally adopted by virtually all PC, workstation and midrange hardware manufactures and tens of billions of such machines have been produced. Now we use the features in this standard to improve the original algorithm.

In this paper, we describe an algorithm in floating-point arithmetic to compute the exact inertia of a real symmetric (shifted) tridiagonal matrix. The inertia, denoted by the integer triplet (π, ν, ζ), is defined as the number of positive, negative and zero eigenvalues of a real symmetric (or complex Hermitian) matrix and the adjective exact refers to the eigenvalues computed in exact arithmetic. This requires the floating-point computation of the diagonal matrix D of the LDL^{t} factorization of the shifted tridiagonal matrix $T - \tau I$ with $+\infty$ and $-\infty$ rounding modes defined in IEEE 754 standard. We are not aware of any other algorithm which gives the exact answer to a numerical problem when implemented in floating-point arithmetic in standard working precisions. The guaranteed intervals for eigenvalues are obtained by bisection or multisection with this exact inertia information. Similarly, using the Golub–Kahan form, guaranteed intervals for singular values of bidiagonal matrices can be computed. The diameter of the eigenvalue (singular value) intervals depends on

* Tel.: +44 1865 287816; fax: +44 1865 287814.
E-mail address: vince@strubi.ox.ac.uk

doi:10.1016/j.laa.2006.09.008

the number of shifts with inconsistent inertia in two rounding modes. Our algorithm not only guarantees the accuracy of the solutions but is also consistent across different IEEE 754 standard compliant architectures. The unprecedented accuracy provided by our algorithms could be also used to debug and validate standard floating-point algorithms for computation of eigenvalues (singular values). Accurate eigenvalues (singular values) are also required by certain algorithms to compute accurate eigenvectors (singular vectors).

We demonstrate the accuracy of our algorithms by using standard matrix examples. For the Wilkinson W_{21}^{+} matrix, the eigenvalues (in IEEE double precision) are very accurate with an (open) interval diameter of 6 ulps (units of the last place held of the mantissa) for one of the eigenvalues and lesser (down to 2 ulps) for others. These results are consistent across many architectures including Intel, AMD, SGI and DEC Alpha. However, by enabling IEEE double extended precision arithmetic in Intel/AMD 32-bit architectures at no extra computational cost, the (open) interval diameters were reduced to one ulp, which is the best possible solution for this problem. We have also computed the eigenvalues of a tridiagonal matrix which manifests in Gauss–Laguerre quadrature and the results are extremely good in double extended precision but less so in double precision. To demonstrate the accuracy of computed singular values, we have also computed the eigenvalues of the Kac_{30} matrix, which is the Golub–Kahan form of a bidiagonal matrix. The tridiagonal matrix has known integer eigenvalues. The bidiagonal Cholesky factor of the Gauss–Laguerre tridiagonal is also included in the singular value study.

Keywords: Bisection; Multisection; Singular values; Eigenvalues; Bidiagonal matrices; Symmetric tridiagonal matrices; Jacobi matrices; The Golub–Kahan form; LDL^{t} factorization; Interval arithmetic; Monotonic arithmetic; Floating-point arithmetic; IEEE arithmetic; Floating-point rounding modes; Parallel algorithms; Numerical quadrature

1. Introduction and summary

This report may be considered as a non-trivial extension of an unpublished report by William Kahan [24]. His interplay between matrix theory and computer arithmetic led to the development of algorithms for computing accurate eigenvalues and singular values. His report is generally considered as the precursor for the development of IEEE standard 754 for binary arithmetic. This standard has been universally adopted by virtually all PC, workstation and midrange hardware manufactures and tens of billions of such machines have been produced. No other mathematical algorithm has influence the world in this way. Now we use the features in this standard to improve the original algorithms.

In this paper, we describe a floating-point algorithm to compute exact inertia of a symmetric tridiagonal (Jacobi) matrix. Inertia of a complex Hermitian or a real symmetric matrix is defined as the number of positive, negative and zero eigenvalues of that matrix [20,27]. The adjective exact refers to eigenvalues computed in infinite precision arithmetic. Computation of inertia is the basic step in bisection and multisection algorithms for determination of eigenvalues of real symmetric tridiagonal matrices. There are reliable algorithms to transform real symmetric matrices and complex Hermitian matrices to the real symmetric tridiagonal format [3,16] and hence the tridiagonal problem is the ultimate step in computing eigenvalues of real symmetric (or complex Hermitian) matrices. There are also many applications where symmetric tridiagonal matrices occur in their own right such as in the derivation of quadrature formulae [17,25].

Modern bisection algorithms for eigenvalue problems [11,24] rely on the Sylvester–Jacobi inertia theorem [20,27] for the determination of inertia of $T - \tau I$, where T is the real symmetric tridiagonal matrix, I is the identity matrix and τ is the shift. This requires the LDL^{t} factorization of the shifted tridiagonal matrix, where the matrix D is diagonal and L^{t} denotes the transpose of the

lower bidiagonal matrix L. The diagonal values of D are called pivots, which are unique if the factorization exists. Bisection type algorithms can also be designed using Sturm sequence properties; see Gantmacher [14] and Wilkinson [28]. In exact arithmetic, Sturm sequences and Sylvester–Jacobi inertia are give equivalent results. However, Sturm sequences, which are given by the principal minors of the shifted matrix, tend to break down in floating-point arithmetic due to overflow, underflow and other floating-point problems but the inertia counts are more robust; see Kahan [24].

Accurate determination of inertia leads to accurate computation of eigenvalues. There are various perturbational theorems and traditional error analyses for real symmetric tridiagonal matrices which, indicate the expected accuracy of the computed eigenvalues in floating-point arithmetic. See in particular, Kahan [24], Barlow and Demmel [2], Demmel and Kahan [7] and Fernando [11]. However, we establish the accuracy of our results using certain arithmetic axioms and not via error analyses and perturbation theory. These arithmetic axioms, which were first elucidated by Kahan [24], conform to the IEEE standard 754 for binary arithmetic [21]. They were used by Kahan [24], Demmel et al. [8] and Fernando [11] to prove that computed inertia counts in floating-point arithmetic are monotonic with respect to the shift τ if the algorithms are properly designed. The same methodology was also used in the validation of the accuracy of eigenvectors; see Fernando [12].

Our algorithm, which is implemented in IEEE arithmetic with $+\infty$ and $-\infty$ rounding modes [21], gives the exact inertia of a shifted real symmetric tridiagonal matrix. This is a rather surprising result since floating-point algorithms in standard working precisions generally do not give the exact (infinite precision arithmetic) answer to a numerical problem; instead good algorithms give backward stable results [28]. The caveat is that it is not possible to compute inertia at every floating-point shift τ and we designate such failed shift values as dead shifts. By knowing the changes in exact inertia between two shift values, it is possible to find exact inclusion regions (open intervals) for eigenvalues. Dead shifts, when present, increase the diameters of computed eigenvalue intervals.

If all or a significant part of the eigenvalues are required then our algorithms are fully parallelizable and they can be used to find all or a selected subset of eigenvalues (singular values). They can be implemented in shared memory and distributed memory machines, without any major difficulties.

Computation of exact inclusion regions is comparatively inexpensive. The first step is to compute the approximate eigenvalues of the matrix using any reliable floating-point algorithm. On serial machines, if the complete (or a significant part of the) spectrum is required, we prefer to use the Pal–Walker–Kahan algorithm [1] to get good initial estimates of the eigenvalues. In parallel environments, the initial estimates can be computed by bisection or multisection. From now on, we do not explicitly mention multisection since all bisection algorithms in this report can be reformulated as multisection.

Once the approximate eigenvalues are known, the pivots (diagonal values of D) are then computed using both the $+\infty$ rounding mode as well as the $-\infty$ rounding mode for a particular value of τ, in the neighbourhood of an approximate eigenvalue. If the two pivot sets, computed using the two rounding modes corresponding to a shift τ, pass a certain consistency test then the inertia given by these pivots are mathematically exact. Failed shifts are inconclusive and widen the eigenvalue inclusion regions.

We also prove that bisection with the IEEE nearest rounding mode provides extremely accurate eigenvalues. In LAPACK [1], ScaLAPACK [4] and elsewhere, the bisection process is terminated when the interval width is less than the product of the machine precision and the one-norm of the matrix. In the context of our problem, it is meaningful to continue the bisection process until there is no further improvement. Specifically, the best eigenvalue interval computed with IEEE

nearest rounding mode is within the guaranteed interval obtain by our primary algorithm if certain conditions are met.

It is also possible to cast our algorithm in the framework of interval arithmetic. Such a program would hide the intricate details and hence would be concise. However, we expect the interval version to be very much slower than our basic implementation, which does not depend on an interval arithmetic package.

Rectangular matrices are usually orthogonally transformed to bidiagonal matrices for singular value computation. There are many ways to transform a bidiagonal singular value problem to a tridiagonal eigenvalue problem [11,16,15] but we use the Golub–Kahan form, which is a zero diagonal symmetric tridiagonal matrix to compute singular values of bidiagonal matrices; the details are presented in the relevant section. In serial machines, we use *dqds* [13] to compute initial singular values of bidiagonal matrices. Alternatively, we compute the eigenvalues of the Golub–Kahan form via the PWK algorithm [1]. In parallel environments, we proceed directly to bisection or multisection.

This paper is organised as follows. After this introductory section, certain preliminary details are presented, including an algorithm to compute pivots of a shifted tridiagonal matrix, in Section 2. Floating point arithmetic axioms required for our analysis are established in Section 3. Algorithms for computation of exact inertia of shifted tridiagonal matrices are developed in Section 4. Section 5 briefly identifies how to implement our algorithms using interval arithmetic. Computation of eigenvalue and singular value intervals are the issues considered in Section 6 and our numerical results are presented in Section 7. Our conclusions are in Section 8.

2. Preliminaries

2.1. Notation

Upper-case Roman letters denote matrices while lower-case Roman and Greek letters denote scalars. The singular values of an n by n real matrix C are arranged in monotone non-increasing order and denoted by $\sigma_1, \sigma_2, \ldots, \sigma_n$; their union is $\sigma[C]$. The eigenvalues of the real symmetric matrix A are ordered in the monotone non-decreasing order; they are denoted by $\lambda_1, \ldots, \lambda_n$ and the union of these eigenvalues is $\lambda[A]$.

We shall be concerned mainly with real upper bidiagonal matrices and real symmetric tridiagonal matrices. The diagonal elements of the bidiagonal matrix B are denoted by a_i and the super-diagonals by b_i. The diagonal and sub-diagonal elements of the real symmetric tridiagonal matrix T are α_i and β_i, respectively,

$$B = \begin{bmatrix} a_1 & b_1 & & & & & \\ & a_2 & b_2 & & & & \\ & & \cdot & \cdot & & & \\ & & & \cdot & \cdot & & \\ & & & & a_{\underline{n}} & b_{\underline{n}} & \\ & & & & & a_n \end{bmatrix}, \quad T = \begin{bmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_1 & \alpha_2 & \cdot & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \alpha_{\underline{n}} & \beta_{\underline{n}} & \\ & & & & \beta_{\underline{n}} & \alpha_n \end{bmatrix},$$

where $\underline{n} = n - 1$. We define z_i, $i = 1, n-1$ as β_i^2.

Without loss of generality, we assume that none of the super-diagonal elements b_i (or β_i) is zero; otherwise the matrix B (or T) splits into two submatrices, which can be then analyzed separately. We also use the *qd* variables $q_i = a_i^2$, and $e_i = b_i^2$. The bold case letter $\mathbf{q}$ denotes the array q_i, $i = 1, \ldots, n$ and similarly $\mathbf{e}$, α and $\mathbf{z}$ represent arrays.

We assume that the arrays **q** and **e** are exactly machine representable and our computed singular values correspond to values defined by the **q** and **e** arrays; they could be marginally different from values computed from the **a** and **b** arrays. Similarly, it is assumed that α and **z** are exactly machine representable and the computed eigenvalues refer to the values defined by the α and **z** arrays.

2.2. Inertia

We make frequent references to the LDL^{t} factorization of a shifted symmetric tridiagonal matrix $T - \tau I$. We use ν to denote the number of negative diagonal elements of D, which according to the Sylvester–Jacobi inertia theorem, gives the number of negative eigenvalues of $T - \tau I$. Similarly, π is the number of positive elements of D and it indicates the number of positive eigenvalues of $T - \tau I$. A simple algorithm exists for computing the diagonal elements of the matrix D [9–11,24].

Lemma 1. *The diagonal elements of the diagonal matrix D of the LDL^{t} factorization of $T - \tau I$ are given by*

$$\begin{aligned} &d_i = \alpha_i - \tau, \quad \text{for } i = 1, \\ &d_i = (\alpha_i - (z_{i-1}/d_{i-1})) - \tau, \quad \text{for } i = 2, \ldots, n. \end{aligned} \tag{1}$$

In Eq. (1), d_i can become infinite if a particular d_{i-1} is zero. In exact arithmetic this can happen only at a finite number of values of τ. There are several ways to avoid this problem in floating-point arithmetic. The traditional approach is to continue the iteration with a perturbed value of d_{i-1}, which may be interpreted as giving a tiny perturbation to α_{i-1}. Since we seek the exact inertia of the matrix and not of a perturbed matrix, we abandon the iteration if any $|d_{i-1}|$ becomes less than a tiny carefully selected normalized floating-point number η and restart the iteration with a different shift τ. We call such a failed τ a dead shift of type 1. If IEEE floating-point arithmetic [21] with signed zeros and infinities is available then it is also possible to continue the iteration in spite of infinite values; see [8] for more details.

The parentheses in (1) are paramount in our analysis and implementation. Without them monotone properties associated with eigenvalue counting break down. Unfortunately, in the LAPACK routine *dstebz* and its counterpart in ScaLAPACK, these parentheses were not included as they were designed for supercomputers of a bygone era. However, even if these parentheses are included, modern compilers with aggressively optimising flags often ignore these parentheses.

We are now ready to present our basic algorithm for inertia determination of a real symmetric tridiagonal matrix. At this stage, it is considered as an algorithm in exact arithmetic. However, with appropriate floating-point rounding, this will be moulded to give our floating-point algorithms. This is an implementation of Lemma 1.

Algorithm 1 (*Pivots of a shifted tridiagonal matrix*).

function $\mathbf{d} = \text{pivots}(\alpha, \mathbf{z}, \tau)$
$d_1 := \alpha_1 - \tau$
if $|d_1| < \eta$ then abort (flag type 1 failure)
for $i = 2, \ldots, n$
  $d_i := (\alpha_i - (z_{i-1}/d_{i-1})) - \tau$
  if $|d_i| < \eta$ and $i \neq n$ then abort (flag type 1 failure)
end for

The positive scalar η is a carefully selected tolerance to avoid overflow in the division operation. However, in exact arithmetic, η could be arbitrarily small.

2.3. Intervals

Let x be a real number or a floating-point number and similarly, y. If $y \geqslant x$ then $[x, y]$ is called a proper interval; Otherwise, it is an improper interval.

If x and y are both positive (negative) then $[x, y]$ is a positive (negative) interval. Positive and negative intervals are called sign definite intervals. In a proper interval, if either x or y is zero then $[x, y]$ is a sign semi-definite interval. Proper intervals, where $\text{sign}(x) = -\text{sign}(y)$ are sign indefinite intervals.

3. Floating-point arithmetic

3.1. Rounding

There are several rounding modes defined in the IEEE Standard 754 [21] for binary floating-point arithmetic. We denote rounding modes in the directions of plus infinity and minus infinity by $\overset{+\infty}{\rightarrow}$ and $\overset{-\infty}{\rightarrow}$, respectively. Rounding to the nearest machine number is indicated by $\leftrightarrow$. By default, if a rounding mode is not indicated then exact arithmetic is assumed.

The value of the sum of the two floating-point numbers a and b using the plus infinity rounding mode will be illustrated as

$$c = \{a + b\}_{\overset{+\infty}{\rightarrow}}.$$

If the same rounding mode is used for evaluating an expression with several arithmetic operations then we denote the rounding mode only once. As an example,

$$\{\{a * b\}_{\overset{+\infty}{\rightarrow}} + \{c - d\}_{\overset{+\infty}{\rightarrow}}\}_{\overset{+\infty}{\rightarrow}} = \{(a * b) + (c - d)\}_{\overset{+\infty}{\rightarrow}}.$$

Let $\circ$ denote any one of the basic arithmetic operations, $+$, $-$, $\times$ or $\div$. We assume that the following axioms are obeyed by floating-point hardware and software. IEEE arithmetic, when implemented correctly in hardware and software, conforms to these rules.

Axiom 1. Let b be a normalized floating-point number. Then unary negation of a gives the exact result in any standard rounding mode.

$$-b = \{-b\}_{\overset{-\infty}{\rightarrow}} = \{-b\}_{\leftrightarrow} = \{-b\}_{\overset{+\infty}{\rightarrow}}.$$

Axiom 2. Let a and b be any two normalized floating-point numbers. Then scalar operations $+$ and $*$ are commutative,

$$\{a + b\}_r = \{b + a\}_r,$$
$$\{a * b\}_r = \{b * a\}_r,$$

where r denotes any standard rounding mode.

Axiom 3. Let a and b be any two normalized floating-point numbers and $a \circ b$ does not underflow (including denormalization), overflow, NaN or infinity on evaluation. If $a \circ b$ is exactly machine representable then

$$a \circ b = \{a \circ b\}_{\overset{-\infty}{\rightarrow}} = \{a \circ b\}_{\leftrightarrow} = \{a \circ b\}_{\overset{+\infty}{\rightarrow}}.$$

If it is not machine representable then one of the following two relationships is true and the other is false.

$$\{a \circ b\}_{\overrightarrow{-\infty}} < \{a \circ b\}_{\leftrightarrow} = \{a \circ b\}_{\overrightarrow{+\infty}},$$
$$\{a \circ b\}_{\overrightarrow{-\infty}} = \{a \circ b\}_{\leftrightarrow} < \{a \circ b\}_{\overrightarrow{+\infty}}.$$

In general,

$$\{a \circ b\}_{\overrightarrow{-\infty}} \leqslant \{a \circ b\}_{\leftrightarrow} \leqslant \{a \circ b\}_{\overrightarrow{+\infty}},$$
$$\{a \circ b\}_{\overrightarrow{-\infty}} \leqslant a \circ b \leqslant \{a \circ b\}_{\overrightarrow{+\infty}}.$$

Definition 1. For any two normalized floating-point numbers a and b if

$$\{a \circ b\}_{r_1} \leqslant \{a \circ b\}_{r_2},$$

where r_1 and r_2 are two IEEE rounding modes (one of them could be exact arithmetic), then we use the symbolic notation $r_1 \leqslant r_2$.

Axiom 4. Let a and b be any two normalized floating-point numbers and $a \circ b$ does not underflow (including denormalization), overflow, NaN or infinity on evaluation. Then for $\circ = *$ or /,

$$\{a \circ b\}_{\overrightarrow{-\infty}} = -\{(-a) \circ b\}_{\overrightarrow{+\infty}},$$
$$\{a \circ b\}_{\overrightarrow{+\infty}} = -\{(-a) \circ b\}_{\overrightarrow{-\infty}}.$$

For $\circ = +$ or $-$,

$$\{a \circ b\}_{\overrightarrow{-\infty}} = -\{(-a) \circ (-b)\}_{\overrightarrow{+\infty}},$$
$$\{a \circ b\}_{\overrightarrow{+\infty}} = -\{(-a) \circ (-b)\}_{\overrightarrow{-\infty}}.$$

The last axiom indicates that the $\overrightarrow{+\infty}$ rounding mode can be simulated using the $\overrightarrow{-\infty}$ rounding mode and vice versa. Thus, it is not necessary to change the rounding mode to implement our algorithms except perhaps initially to exit from the default (nearest number) rounding mode. It is well known that it is expensive to dynamically change rounding modes in some current machine architectures.

3.2. Monotonic arithmetic

The integer $\nu(\tau)$ counts the eigenvalues of T which are less than τ. So by definition, ν is monotone increasing in τ, but that is in the context of exact arithmetic. In floating-point arithmetic, the computed values of ν are also monotonic in τ provided that the arithmetic unit obeys certain reasonable axioms. Such arithmetic units are said to provide monotonic arithmetic operations (see Kahan [24]). Similarly, $\pi(\tau)$ counts the eigenvalues of T, which are greater than τ.

Axiom 5. Let $\{\cdot\}$ denote a floating-point operation which does not lead to underflow (including denormalization), overflow, NaN or infinity in the evaluation of the result. Let r_1 and r_2 be two rounding modes and $r_2 \geqslant r_1$. For any machine representable normalized numbers a, b, and c,

(+) if $a \leqslant \tilde{a}$ and $\{a + b\}_{r_1} > \{\tilde{a} + c\}_{r_2}$ then $b > c$,
(−) if $a \leqslant \tilde{a}$ and $\{a - b\}_{r_1} > \{\tilde{a} - c\}_{r_2}$ then $b < c$,
(×) if $a > 0$ and $\{a * b\}_{r_1} > \{a * c\}_{r_2}$ then $b > c$,
(÷) if $a > 0$, $b \neq 0$, $c \neq 0$ and $\left\{\frac{a}{b}\right\}_{r_1} > \left\{\frac{a}{c}\right\}_{r_2}$ then

$c > b$ if sign (b) = sign(c),

$b > 0$, $c < 0$ if sign (b) = −sign(c).

4. Exact inertia of tridiagonal matrices

We use two algorithms to compute the pivots of the LDL^{t} factorization of $T - \tau I$ using both $\overset{+\infty}{\rightarrow}$ and $\overset{-\infty}{\rightarrow}$ roundings. The following two algorithms for computing pivots using floating-point arithmetic are identical to Algorithm 1 except for the inclusion of floating-point rounding modes. Before we display our algorithms, we identify two types of failure.

Definition 2 (*type* 1 *and* 2 *failures*).

1. If a pivot (d_i^+ or d_i^-) is less than η (as defined in Section 2.2) then it is a type 1 failure.
2. If $\operatorname{sign}(d_i^+) \neq \operatorname{sign}(d_i^-)$ then it is a type 2 failure.

Algorithm 2 (*Pivot set* $\mathbf{d}^+$ *of a shifted tridiagonal matrix*).

function $\mathbf{d}^+ = \text{pivots}^+(\alpha, \mathbf{z}, \tau)$
$d_1^+ := \{\alpha_i - \tau\}_{\overset{+\infty}{\rightarrow}}$
if $|d_1^+| < \eta$ then abort (flag type 1 failure)
for $i = 2, \ldots, n$

$$d_i^+ := \{(\alpha_i - \{z_{i-1}/d_{i-1}^+\}_{\overset{-\infty}{\rightarrow}}) - \tau\}_{\overset{+\infty}{\rightarrow}} \tag{2}$$

if $|d_i^+| < \eta$ and $i \neq n$ then abort (flag type 1 failure)

end for

Algorithm 3 (*Pivot set* $\mathbf{d}^-$ *of a tridiagonal matrix*).

function $\mathbf{d}^- = \text{pivots}^-(\alpha, \mathbf{z}, \tau)$
$d_1^- := \{\alpha_i - \tau\}_{\overset{-\infty}{\rightarrow}}$
if $|d_1^-| < \eta$ then abort (flag type 1 failure)
for $i = 2, \ldots, n$

$$d_i^- := \{(\alpha_i - \{z_{i-1}/d_{i-1}^-\}_{\overset{+\infty}{\rightarrow}}) - \tau\}_{\overset{-\infty}{\rightarrow}} \tag{3}$$

if $|d_i^-| < \eta$ and $i \neq n$ then abort (flag type 1 failure)

end for

It is possible to transform the above two algorithms to a single algorithm except for the rounding mode.

Lemma 2. *Let* $\hat{z}_i = -z_i = -b_i^2$. *Then Eqs.* (2) *and* (3) *can be replaced by*

$$d_i^+ := \{(\alpha_i + (\hat{z}_{i-1}/d_{i-1}^+)) - \tau\}_{\overset{+\infty}{\rightarrow}},$$

$$d_i^- := \{(\alpha_i + (\hat{z}_{i-1}/d_{i-1}^-)) - \tau\}_{\overset{-\infty}{\rightarrow}}.$$

Proof. This is a direct application of Axiom 4. □

We have already seen that when confined to a basic arithmetic operation, the result in $\overset{+\infty}{\rightarrow}$ rounding mode gives a result which is not smaller than the answer in the $\overset{-\infty}{\rightarrow}$ mode. Thus, it is

trivial to show that $d_1^+(\tau)$ is not smaller than $d_1^-(\tau)$ for any τ. It is our intention to show that $d_i^+(\tau)$ is not smaller than $d_i^-(\tau)$ for $i > 1$ as long as certain conditions are met; that is, $[d_i^-(\tau), d_i^+(\tau)]$ is a proper closed interval under favourable conditions. First we investigate possible failures of this proposition.

Lemma 3. *Let the pair* $\left\{\delta_i^{(1)}(\tau), \delta_i^{(2)}(\tau)\right\}$ *denote any one instantiation of the following pivot combinations*:

$$\begin{aligned}\left\{\delta_i^{(1)}(\tau), \delta_i^{(2)}(\tau)\right\} &= \{d_i^-(\tau), d_i^+(\tau)\},\\ \left\{\delta_i^{(1)}(\tau), \delta_i^{(2)}(\tau)\right\} &= \{d_i^-(\tau), d_i(\tau)\},\\ \left\{\delta_i^{(1)}(\tau), \delta_i^{(2)}(\tau)\right\} &= \{d_i(\tau), d_i^+(\tau)\},\end{aligned} \tag{4}$$

where $d_i(\tau)$ *denotes the pivots in exact arithmetic. Suppose that* $\delta_k^{(2)}(\tau) < \delta_k^{(1)}(\tau)$, $k > 1$, *for a particular* τ. *Then either*

$$\begin{aligned}&\delta_{k-1}^{(2)}(\tau) < \delta_{k-1}^{(1)}(\tau) \text{ or}\\ &\delta_{k-1}^{(2)}(\tau) > 0, \delta_{k-1}^{(1)}(\tau) < 0\end{aligned}$$

is true provided no underflow (*including denormalization*), *overflow*, *NaN or infinity on evaluation in any one of the relevant algorithms*.

Proof. We supply a proof for the case defined by (4),

$$d_k^+(\tau) < d_k^-(\tau);$$

the other two cases follow the same methodology. Since

$$\begin{aligned}d_k^+(\tau) &= \{(\alpha_k - \{z_{k-1}/d_{k-1}^+(\tau)\}_{\to}^{-\infty}) + (-\tau)\}_{\to}^{+\infty},\\ d_k^-(\tau) &= \{(\alpha_k - \{z_{k-1}/d_{k-1}^-(\tau)\}_{\to}^{+\infty}) + (-\tau)\}_{\to}^{-\infty},\end{aligned}$$

(see Algorithms 2 and 3), we have

$$\{(\alpha_k - \{z_{k-1}/d_{k-1}^-(\tau)\}_{\to}^{+\infty}) + (-\tau)\}_{\to}^{-\infty} > \{(\alpha_k - \{z_{k-1}/d_{k-1}^+(\tau)\}_{\to}^{-\infty}) + (-\tau)\}_{\to}^{+\infty},$$

Using the rule (+) of Axiom 5, we get

$$\{(\alpha_k - \{z_{k-1}/d_{k-1}^-(\tau)\}_{\to}^{+\infty})\}_{\to}^{-\infty} > \{(\alpha_k - \{z_{k-1}/d_{k-1}^+(\tau)\}_{\to}^{-\infty})\}_{\to}^{+\infty},$$

Applying rule (−) of the same axiom, we obtain

$$\{z_{k-1}/d_{k-1}^-(\tau)\}_{\to}^{+\infty} < \{z_{k-1}/d_{k-1}^+(\tau)\}_{\to}^{-\infty}$$

From rule (÷) in Axiom 5, if $d_{k-1}^-(\tau)$ and $d_{k-1}^+(\tau)$ have the same sign then we get the first condition,

$$d_{k-1}^+(\tau) < d_{k-1}^-(\tau).$$

If the signs are different then we obtain the second condition,

$$d_{k-1}^{+}(\tau) > 0, \quad d_{k-1}^{-}(\tau) < 0. \qquad \square$$

The above result indicates that an improper pivot pair $\{d_k^-(\tau), d_k^+(\tau)\}$, where $d_k^+(\tau) < d_k^-(\tau)$, is either generated by an improper pair $\{d_{k-1}^-(\tau), d_{k-1}^+(\tau)\}$, where $d_{k-1}^+(\tau) < d_{k-1}^-(\tau)$ or by a sign indefinite interval $[d_{k-1}^-(\tau), d_{k-1}^+(\tau)]$. The following lemma further sharpens this result.

Lemma 4. *Using the notation established in Lemma 3, suppose that* $\delta_k^{(2)}(\tau) < \delta_k^{(1)}(\tau)$ *for an index* $k, k > 1$. *Then an index* j, *in the range* $2, \ldots, k-1$, *exists such that* $\delta_j^{(2)}(\tau) > 0$ *and* $\delta_j^{(1)}(\tau) < 0$.

Proof. Again we provide a proof only for the case defined by (4) but the other two cases follow the same concept.

We recursively apply Lemma 3 one or more times until we arrive at a j, $j = 1, \ldots, k-1$, where $d_j^+(\tau) > 0$ and $d_j^-(\tau) < 0$. However, $d_1^+(\tau) \geqslant d_1^-(\tau)$ and hence $j \neq 1$. $\square$

The previous lemma shows that the root cause of an improper pivot is an earlier sign indefinite pivot. Thus, once a sign indefinite pivot is discovered, we terminate the algorithm. The failed shift is called a dead shift of type 2. Note that sign semi-definite pivots are avoided by dead shifts of type 1.

The following theorem is one of our primary results; it shows that exact inertia of a real symmetric tridiagonal matrix can be computed using floating-point arithmetic.

Theorem 1. *Suppose that the pivots* $\{d_i^-(\tau), d_i^+(\tau)\}$, $i = 1, \ldots, n$ *are sign definite for a particular* τ. *Then, for* $i = 1, \ldots, n$

$$d_i^-(\tau) \leqslant d_i(\tau) \leqslant d_i^+(\tau),$$

where $d_i(\tau)$ *are the diagonal pivots as computed using exact arithmetic* (*see Algorithm 1*).

Proof. The proof is by contradiction. Suppose that $d_i(\tau)$ is in the closed interval $[d_i^-(\tau), d_i^+(\tau)]$, for $i = 1, \ldots, k-1$. This is always true for $i = 1$. For an index $k > 1$, if $d_k(\tau)$ is not in the interval $[d_k^-(\tau), d_k^+(\tau)]$, then either $d_k^+(\tau) < d_k(\tau)$ or $d_k^-(\tau) > d_k(\tau)$ but not both.

We first consider the possibility $d_k^+(\tau) < d_k(\tau)$. Thus, from Lemma 4, an index j, $j = 2, \ldots, k-1$ exists such that $d_j^+(\tau) > 0$ and $d_j(\tau) < 0$. However, this contradicts the assumption that the intervals $[d_i^-(\tau), d_i^+(\tau)]$, are sign definite and hence $d_i(\tau)$ is within this closed interval for $i = 1, \ldots, k-1$.

The second possibility, $d_k^-(\tau) > d_k(\tau)$ also leads to a similar contradictory conclusion. Thus, $d_k(\tau)$ is in the closed interval $[d_k^-(\tau), d_k^+(\tau)]$ as claimed. $\square$

The previous theorem can be utilized to design an algorithm for the exact determination of inertia of a real symmetric tridiagonal matrix.

Algorithm 4 (*Exact inertia of a tridiagonal matrix*).

function $(\nu, \pi) = \text{inertia}(\alpha, \mathbf{z}, \tau)$

$\mathbf{d}^+ = \text{inertia}^+(\alpha, \mathbf{z}, \tau)$

$\mathbf{d}^- = \text{inertia}^-(\alpha, \mathbf{z}, \tau)$
for $i = 1, \ldots, n$
  if sign $(d_i^+) = \text{sign}(d_i^-)$ then
    if $d_i^+ < 0$ then
      $\nu := \nu + 1$
    else if $d_i^+ > 0$ then
      $\pi := \pi + 1$
    end if
  else
    abort (flag type 2 failure)
  end if
end do

Since the usual default is the nearest number rounding mode, it is intriguing to find out the accuracy of inertia in this mode.

Theorem 2. *Suppose that the pivots* $[d_i^-(\tau), d_i^+(\tau)]$, $i = 1, \ldots, n$ *are sign definite for a particular* τ. *Then, for* $i = 1, \ldots, n$

$$d_i^-(\tau) \leqslant d_i^{\leftrightarrow}(\tau) \leqslant d_i^+(\tau),$$

where $d_i^{\leftrightarrow}(\tau)$ *are the diagonal pivots as computed using nearest number rounding.*

Proof. The proof is almost identical to the proof of Theorem 1 with $d_i(\tau)$ replaced by $d_i^{\leftrightarrow}(\tau)$. □

5. Inertia using interval arithmetic

It is feasible to implement our algorithms using basic interval arithmetic operations; see for example Hayes [18]. In particular, we can avoid problems associated with interval division by avoiding sign indefinite interval divisors, which is completely consistent with our theoretical developments. However, we note that there are more complete interval arithmetic systems, where division by sign indefinite intervals is allowed; see for example Hickey et al. [19] and the references therein. Perhaps, such an elaborate interval arithmetic system could provide more optimal interval bounds. There have been at least two attempts to compute eigenvalues of interval symmetric tridiagonal matrices; see Commercon [5] and Pavec [26].

It is possible to combine Algorithms 2 and 3 to give a concise interval algorithm to compute pivots. We assume that the interval arithmetic package automatically takes care of the interval bounds. There is a failure mode for this algorithm.

Definition 3 (*Type* 3 *failure*). If the pivot interval $[d_i^-, d_i^+]$ intersects $[-\eta, \eta]$ then it is a type 3 failure.

Algorithm 5 (*Interval pivots of a tridiagonal matrix*).
  function $(\mathbf{d}^-, \mathbf{d}^+) = \text{intervalPivots}(\alpha, \mathbf{z}, \tau)$
  $[d_1^-, d_1^+] := [\alpha_i, \alpha_i] - [\tau, \tau]$

if $[d_1^-, d_1^+] \cap [-\eta, \eta]$ then abort (flag type 3 failure)
for $i = 2, \ldots, n$
$[d_i^-, d_i^+] := ([\alpha_i, \alpha_i] - ([z_{i-1}, z_{i-1}]/[d_{i-1}^-, d_{i-1}^+])) - [\tau, \tau]$
if $[d_i^-, d_i^+] \cap [-\eta, \eta]$ and $i \neq n$ then abort (flag type 3 failure)
end for

We expect the implementations based on Algorithms 2 and 3 to be more efficient compared with the interval algorithm using an interval arithmetic package. An interval division operation requires four standard divisions while our algorithm requires only two divisions and also we do not have any overheads due to the interval package being used. Furthermore, interval algorithms have to be executed from scratch while our algorithms start from good approximations.

We emphasize that all our displayed algorithms are far away from practical codes as they were designed to emphasize the mathematics of the algorithms. In particular, instead of having conditionals to count inertia, they should be extracted from the sign bit of the pivots. Furthermore, shift failures should be tested outside the inner loop using IEEE arithmetic or otherwise. These features were partially incorporated in our practical implementations.

6. Computing open intervals

6.1. Eigenvalues

Standard bisection algorithms for computation of eigenvalues are based on computing either the inertia count $\nu(\tau)$ or $\pi(\tau)$ and not both for a particular τ. However, in interval computation, there are four inertia counts; $\nu^+(\tau)$ and $\pi^+(\tau)$ based on d_i^+ pivots and ν^- and π^- based on d_i^- pivots. Any discrepancy between these four inertia counts indicates that there is an index j such $\{d_j^-(\tau), d_j^+(\tau)\}$ is not a sign definite interval and hence flagged as type 1 or type 2 error.

We first compute approximate eigenvalues of the tridiagonal matrix using a standard backward stable eigenvalue routine. In the neighbourhood of an approximate eigenvalue, we initiate bisection with sign definite intervals. Since the four inertia counts $\nu^-(\tau)$, $\pi^-(\tau)$, $\nu^+(\tau)$ and $\pi^+(\tau)$ are consistent for sign definite pairs $\{d_i^-(\tau), d_i^+(\tau)\}$, we have to consider only one of them, say $\nu^-(\tau)$.

Suppose that $\nu^-(\tau_1) = k$ and $\nu^-(\tau_2) = k + 1$. Then there is one and only one eigenvalue in the open interval (τ_1, τ_2). If τ_2 is the next machine representable number from τ_1 in the direction of $+\infty$ then there is an eigenvalue unrepresentable in the floating point format in the open interval (τ_1, τ_2).

Often, τ_1 and τ_2 are not two consecutive machine numbers but with intermediate (dead) shifts τ, which are designated as failures. If there are no failures for intermediate shifts, we reduce the interval (τ_1, τ_2) using bisection.

6.2. Singular values

Interval computation of singular values is implemented by transforming the singular value problem to an equivalent eigenvalue problem. There are many ways to convert a singular value problem to an equivalent eigenvalue problem; see Section 8.3 of Golub and Van Loan [16], Golub and Kahan [15] and Fernando [11]. We define a $2n$ by $2n$ symmetric matrix,

$$A = \begin{bmatrix} 0 & C \\ C^{\mathrm{t}} & 0 \end{bmatrix}.$$

The eigenvalues of A are then given by the union of the singular values of C and the negated singular values of C, $\lambda[A] = \{\sigma[C]\} \cup \{-\sigma[C]\}$ as presented in Theorem 4.2 of Stewart and Sun [27]. This property was first exploited by Jordan [23].

If C is bidiagonal (that is, $C = B$), then Golub and Kahan [15] discovered that A can be condensed to a tridiagonal matrix with zero diagonal entries by using a permutational similarity transformation equivalent to a perfect shuffle. This tridiagonal matrix, which we denote by T_0 is called the Golub–Kahan form.

$$T_0 = \begin{bmatrix} 0 & a_1 & & & & & \\ a_1 & 0 & b_1 & & & & \\ & b_1 & 0 & a_2 & & & \\ & & a_2 & 0 & . & & \\ & & & . & 0 & . & \\ & & & & . & 0 & a_n \\ & & & & & a_n & 0 \end{bmatrix}.$$

7. Numerical examples

7.1. Architectures

We have used three sets of architectures in our study. The first is the ubiquitous Intel and AMD 32-bit processors, which has the option to run either in IEEE double precision or IEEE double extended precision. The second set is the dated 64-bit processors such as the SGI MIPS and DEC Alpha machines. The third is the modern 64-bit processors from AMD and Intel. The details of these machines are available in Table 1.

Table 1
Test architectures, operating systems and compilers

Architecture	Bits	Operating System	Compiler
Intel Xeon dual 2.66 GHz	32	Red Hat Linux 2.4.21-4.ELsmp	gcc 3.2.2
Intel Pentium M 1.3 GHz	32	Fedora 1.0 2.4.22-1.2115.nptl	gcc 3.3.2
Intel Pentium M 1.7 GHz	32	Fedora 4.0 2.6.11-1.1286-FC4	gcc 4.0.0
Intel Pentium 3 (mark 1) 500 MHz	32	Mandrake Linux 2.96-0.48	gcc 3.3.2
AMD Athlon (Thunderbird) 800 MHz	32	Fedora 1.0 Linux	gcc 2.96
AMD Athlon XP1700+ (Thoroughbred) 1.47 GHz	32	Fedora 3.0 2.6.11-1.14-FC3	gcc 3.4.3
SGI MIPS IP27 360 MHz	64	IRIX64 6.5	MIPSpro 7.30
Alpha EV6.8AL 833 MHz	64	OSF1 5.1	DEC Fortran V5.4-1283
AMD Opteron 246 2.0 GHz dual	64	SUSE Linux 2.6.114-21.8-smp	gcc 3.3.5
Intel Xeon 3.2 GHz	64	Red Hat Linux 2.4.21-32.EL	gcc 3.2.3

On machines, which have the Gnu gcc/glibc compiler, the rounding modes were selected using *fesetround*, which is defined in the C99 language standard [22]. On SGI and DEC Alpha machines, vendor supplied routines were used to change the rounding modes.

7.2. Eigenvalues

In all our examples, we have scaled the matrices by 2^{256} to reduce the chances of underflow. For this value and higher values, underflow was rare. The value of η (see Section 2.2) was chosen as the smallest normalized positive number in double precision, 2^{-1023}.

Table 2
Inertia near the first 10 eigenvalues of W_{21}^{+} in double precision

i	τ	τ hex end	inertia^{-} $\nu^{-}, \pi^{-}, \zeta^{-}$	inertia$^{\leftrightarrow}$ $\nu^{\leftrightarrow}, \pi^{\leftrightarrow}, \zeta^{\leftrightarrow}$	inertia^{+} $\nu^{+}, \pi^{+}, \zeta^{+}$	Fail
1	−1.125441522119984494	0D7E	(0, 21, 0)	(0, 21, 0)	(0, 21, 0)	
1	−1.125441522119984272	0D7D	(1, 20, 0)	(0, 21, 0)	(0, 21, 0)	2
1	−1.125441522119984050	0D7C	(1, 20, 0)	(1, 20, 0)	(1, 20, 0)	
2	0.253805817096677988	BF1D	(1, 20, 0)	(1, 20, 0)	(1, 20, 0)	
2	0.253805817096678044	BF1E	(0, 20, 1)	(1, 20, 0)	(1, 20, 0)	1
2	0.253805817096678099	BF1F	(2, 19, 0)	(1, 20, 0)	(1, 20, 0)	2
2	0.253805817096678155	BF20	(2, 19, 0)	(1, 20, 0)	(1, 20, 0)	2
2	0.253805817096678210	BF21	(2, 19, 0)	(1, 20, 0)	(1, 20, 0)	2
2	0.253805817096678266	BF22	(2, 19, 0)	(2, 18, 1)	(2, 18, 1)	1
2	0.253805817096678321	BF23	(2, 19, 0)	(2, 19, 0)	(2, 19, 0)	
3	0.947534367529293098	FABE	(2, 19, 0)	(2, 19, 0)	(2, 19, 0)	
3	0.947534367529293209	FABF	(3, 18, 0)	(2, 19, 0)	(2, 19, 0)	2
3	0.947534367529293320	FAC0	(3, 18, 0)	(3, 18, 0)	(2, 19, 0)	2
3	0.947534367529293431	FAC1	(3, 18, 0)	(3, 18, 0)	(2, 19, 0)	2
3	0.947534367529293542	FAC2	(3, 18, 0)	(3, 18, 0)	(3, 18, 0)	
4	1.789321352695081080	C1F7	(3, 18, 0)	(3, 18, 0)	(3, 18, 0)	
4	1.789321352695081302	C1F8	(4, 17, 0)	(3, 17, 1)	(3, 18, 0)	1
4	1.789321352695081524	C1F9	(4, 17, 0)	(4, 17, 0)	(3, 18, 0)	2
4	1.789321352695081746	C1FA	(4, 17, 0)	(4, 17, 0)	(4, 17, 0)	
5	2.130209219362505735	65B0	(4, 17, 0)	(4, 17, 0)	(4, 17, 0)	
5	2.130209219362506179	65B1	(5, 16, 0)	(5, 16, 0)	(4, 17, 0)	2
5	2.130209219362506623	65B2	(5, 16, 0)	(5, 16, 0)	(5, 16, 0)	
6	2.961058884185726381	DD13	(5, 16, 0)	(5, 16, 0)	(5, 16, 0)	
6	2.961058884185726825	DD14	(6, 15, 0)	(6, 15, 0)	(5, 16, 0)	2
6	2.961058884185727269	DD15	(6, 15, 0)	(6, 15, 0)	(6, 15, 0)	
7	3.043099292578823167	DAB0	(6, 15, 0)	(6, 15, 0)	(6, 15, 0)	
7	3.043099292578823611	DAB1	(7, 14, 0)	(6, 15, 0)	(6, 15, 0)	2
7	3.043099292578824056	DAB2	(7, 14, 0)	(7, 14, 0)	(7, 14, 0)	
8	3.996048201383624487	7554	(7, 14, 0)	(7, 14, 0)	(7, 14, 0)	
8	3.996048201383624932	7555	(8, 13, 0)	(8, 13, 0)	(7, 14, 0)	2
8	3.996048201383625376	7556	(8, 13, 0)	(8, 13, 0)	(8, 13, 0)	
9	4.004354023440855670	2352	(8, 13, 0)	(8, 13, 0)	(8, 13, 0)	
9	4.004354023440856558	2353	(9, 12, 0)	(8, 13, 0)	(8, 13, 0)	2
9	4.004354023440857446	2354	(9, 12, 0)	(9, 12, 0)	(9, 12, 0)	
10	4.999782477742900966	1414	(9, 12, 0)	(9, 12, 0)	(9, 12, 0)	
10	4.999782477742901854	1415	(10, 11, 0)	(10, 11, 0)	(9, 12, 0)	2
10	4.999782477742902742	1416	(10, 11, 0)	(10, 11, 0)	(10, 11, 0)	

7.2.1. *Wilkinson W_{21}^{+} matrix*

To exhibit the accuracy of our algorithms we have used the well known Wilkinson W_{21}^{+} matrix [28] as an example problem. Our programs were executed in IEEE double precision on 64-bit machines which have 53 bits of precision. These programs were also run in Intel and AMD 32-bit machines in IEEE double precision and IEEE double extended precision. In double extended precision arithmetic, the precision is 64 bits.

In double precision, the legacy 64-bit processors (SGI MIPS and DEC Alpha) and all AMD and Intel 32-bit and 64-bit processors gave identical answers for this W_{21}^{+} eigenvalue problem. There were three eigenvalues with interval diameters of 6, 4 and 3 but all other diameters were 2. Thus, we have been able to compute eigenvalues of this matrix very accurately using floating point arithmetic.

However, the results for Intel and AMD 32-bit machines in double extended precision are very good and optimal; all the diameters are equal to unity. The new Intel and AMD machines 64-bit

Table 3
Inertia near the last 11 eigenvalues of W_{21}^{+} in double precision

i	τ	τ hex end	inertia^{-} $\nu^{-}, \pi^{-}, \zeta^{-}$	inertia$^{\leftrightarrow}$ $\nu^{\leftrightarrow}, \pi^{\leftrightarrow}, \zeta^{\leftrightarrow}$	inertia^{+} $\nu^{+}, \pi^{+}, \zeta^{+}$	Fail
11	5.000244425001912241	8EE3	(10, 11, 0)	(10, 11, 0)	(10, 11, 0)	
11	5.000244425001913129	8EE4	(11, 10, 0)	(11, 10, 0)	(10, 11, 0)	2
11	5.000244425001914017	8EE5	(11, 10, 0)	(11, 10, 0)	(11, 10, 0)	
12	6.000217522257097258	EBEA	(11, 10, 0)	(11, 10, 0)	(11, 10, 0)	
12	6.000217522257098146	EBEB	(12, 9, 0)	(11, 10, 0)	(11, 10, 0)	2
12	6.000217522257099034	EBEC	(12, 9, 0)	(12, 9, 0)	(12, 9, 0)	
13	6.000234031584166239	003B	(12, 9, 0)	(12, 9, 0)	(12, 9, 0)	
13	6.000234031584167127	003C	(13, 8, 0)	(13, 8, 0)	(12, 9, 0)	2
13	6.000234031584168015	003D	(13, 8, 0)	(13, 8, 0)	(13, 8, 0)	
14	7.003951798616373736	4554	(13, 8, 0)	(13, 8, 0)	(13, 8, 0)	
14	7.003951798616374624	4555	(14, 7, 0)	(13, 8, 0)	(13, 8, 0)	2
14	7.003951798616375513	4556	(14, 7, 0)	(14, 7, 0)	(14, 7, 0)	
15	7.003952209528674366	B0BD	(14, 7, 0)	(14, 7, 0)	(14, 7, 0)	
15	7.003952209528675255	B0BE	(15, 6, 0)	(14, 7, 0)	(14, 7, 0)	2
15	7.003952209528676143	B0BF	(15, 6, 0)	(15, 6, 0)	(15, 6, 0)	
16	8.038941115814271399	C8BA	(15, 6, 0)	(15, 6, 0)	(15, 6, 0)	
16	8.038941115814273175	C8BB	(16, 5, 0)	(15, 6, 0)	(15, 6, 0)	2
16	8.038941115814274951	C8BC	(16, 5, 0)	(16, 5, 0)	(16, 5, 0)	
17	8.038941122829021069	0A53	(16, 5, 0)	(16, 5, 0)	(16, 5, 0)	
17	8.038941122829022845	0A54	(17, 4, 0)	(16, 5, 0)	(16, 5, 0)	2
17	8.038941122829024621	0A55	(17, 4, 0)	(17, 4, 0)	(17, 4, 0)	
18	9.210678647304916922	47C0	(17, 4, 0)	(17, 4, 0)	(17, 4, 0)	
18	9.210678647304918698	47C1	(18, 3, 0)	(17, 4, 0)	(17, 4, 0)	2
18	9.210678647304920474	47C2	(18, 3, 0)	(18, 3, 0)	(18, 3, 0)	
19	9.210678647361330462	C3CE	(18, 3, 0)	(18, 3, 0)	(18, 3, 0)	
19	9.210678647361332239	C3CF	(19, 2, 0)	(18, 2, 1)	(18, 3, 0)	2
19	9.210678647361334015	C3D0	(19, 2, 0)	(19, 2, 0)	(19, 2, 0)	
20	10.746194182903320069	1206	(19, 2, 0)	(19, 2, 0)	(19, 2, 0)	
20	10.746194182903321845	1207	(20, 1, 0)	(19, 1, 1)	(19, 2, 0)	2
20	10.746194182903323622	1208	(20, 1, 0)	(20, 1, 0)	(20, 1, 0)	
21	10.746194182903391123	122E	(20, 1, 0)	(20, 1, 0)	(20, 1, 0)	
21	10.746194182903392900	122F	(20, 0, 1)	(20, 1, 0)	(20, 1, 0)	2
21	10.746194182903394676	1230	(21, 0, 0)	(21, 0, 0)	(21, 0, 0)	

machines do not support the IEEE double extended precision and hence they were executed in double precision. These diameters are displayed in Table 5 for all the architectures we have tested. The eigenvalue intervals together with all six corresponding inertia values for all architectures in double precision are tabulated in Tables 2 and 3. For the AMD/Intel machines with double extended precision, the results are shown in Table 4. Since the diameters are 1 ulp, the Wilkinson W_{21}^{+} matrix does not have any machine representable eigenvalues. Overall, the best architecture

Table 4
Inertia near the eigenvalues of W_{21}^{+} in double extended precision

i	τ	τ hex end	inertia^{-} $\nu^{-}, \pi^{-}, \zeta^{-}$	inertia$^{\leftrightarrow}$ $\nu^{\leftrightarrow}, \pi^{\leftrightarrow}, \zeta^{\leftrightarrow}$	inertia^{+} $\nu^{+}, \pi^{+}, \zeta^{+}$	Fail
1	−1.125441522119984272	0D7D	(0, 21, 0)	(0, 21, 0)	(0, 21, 0)	
1	−1.125441522119984050	0D7C	(1, 20, 0)	(1, 20, 0)	(1, 20, 0)	
2	0.253805817096678155	BF20	(1, 20, 0)	(1, 20, 0)	(1, 20, 0)	
2	0.253805817096678210	BF21	(2, 19, 0)	(2, 19, 0)	(2, 19, 0)	
3	0.947534367529293209	FABF	(2, 19, 0)	(2, 19, 0)	(2, 19, 0)	
3	0.947534367529293320	FAC0	(3, 18, 0)	(3, 18, 0)	(3, 18, 0)	
4	1.789321352695081302	C1F8	(3, 18, 0)	(3, 18, 0)	(3, 18, 0)	
4	1.789321352695081524	C1F9	(4, 17, 0)	(4, 17, 0)	(4, 17, 0)	
5	2.130209219362505735	65B0	(4, 17, 0)	(4, 17, 0)	(4, 17, 0)	
5	2.130209219362506179	65B1	(5, 16, 0)	(5, 16, 0)	(5, 16, 0)	
6	2.961058884185726381	DD13	(5, 16, 0)	(5, 16, 0)	(5, 16, 0)	
6	2.961058884185726825	DD14	(6, 15, 0)	(6, 15, 0)	(6, 15, 0)	
7	3.043099292578823611	DAB1	(6, 15, 0)	(6, 15, 0)	(6, 15, 0)	
7	3.043099292578824056	DAB2	(7, 14, 0)	(7, 14, 0)	(7, 14, 0)	
8	3.996048201383624932	7555	(7, 14, 0)	(7, 14, 0)	(7, 14, 0)	
8	3.996048201383625376	7556	(8, 13, 0)	(8, 13, 0)	(8, 13, 0)	
9	4.004354023440856558	2353	(8, 13, 0)	(8, 13, 0)	(8, 13, 0)	
9	4.004354023440857446	2354	(9, 12, 0)	(9, 12, 0)	(9, 12, 0)	
10	4.999782477742901854	1415	(9, 12, 0)	(9, 12, 0)	(9, 12, 0)	
10	4.999782477742902742	1416	(10, 11, 0)	(10, 11, 0)	(10, 11, 0)	
11	5.000244425001912241	8EE3	(10, 11, 0)	(10, 11, 0)	(10, 11, 0)	
11	5.000244425001913129	8EE4	(11, 10, 0)	(11, 10, 0)	(11, 10, 0)	
12	6.000217522257097258	EBEA	(11, 10, 0)	(11, 10, 0)	(11, 10, 0)	
12	6.000217522257098146	EBEB	(12, 9, 0)	(12, 9, 0)	(12, 9, 0)	
13	6.000234031584166239	003B	(12, 9, 0)	(12, 9, 0)	(12, 9, 0)	
13	6.000234031584167127	003C	(13, 8, 0)	(13, 8, 0)	(13, 8, 0)	
14	7.003951798616374624	4555	(13, 8, 0)	(13, 8, 0)	(13, 8, 0)	
14	7.003951798616375513	4556	(14, 7, 0)	(14, 7, 0)	(14, 7, 0)	
15	7.003952209528675255	B0BE	(14, 7, 0)	(14, 7, 0)	(14, 7, 0)	
15	7.003952209528676143	B0BF	(15, 6, 0)	(15, 6, 0)	(15, 6, 0)	
16	8.038941115814273175	C8BB	(15, 6, 0)	(15, 6, 0)	(15, 6, 0)	
16	8.038941115814274951	C8BC	(16, 5, 0)	(16, 5, 0)	(16, 5, 0)	
17	8.038941122829022845	0A54	(16, 5, 0)	(16, 5, 0)	(16, 5, 0)	
17	8.038941122829024621	0A55	(17, 4, 0)	(17, 4, 0)	(17, 4, 0)	
18	9.210678647304916922	47C0	(17, 4, 0)	(17, 4, 0)	(17, 4, 0)	
18	9.210678647304918698	47C1	(18, 3, 0)	(18, 3, 0)	(18, 3, 0)	
19	9.210678647361330462	C3CE	(18, 3, 0)	(18, 3, 0)	(18, 3, 0)	
19	9.210678647361332239	C3CF	(19, 2, 0)	(19, 2, 0)	(19, 2, 0)	
20	10.746194182903320069	1206	(19, 2, 0)	(19, 2, 0)	(19, 2, 0)	
20	10.746194182903321845	1207	(20, 1, 0)	(20, 1, 0)	(20, 1, 0)	
21	10.746194182903392900	122F	(20, 1, 0)	(20, 1, 0)	(20, 1, 0)	
21	10.746194182903394676	1230	(21, 0, 0)	(21, 0, 0)	(21, 0, 0)	

Table 5
Interval diameters of the eigenvalues of W_{21}^{+}

Eigenvalue	All machines double	Intel/AMD 32-bit double extended
1	2	1
2	6	1
3	4	1
4	3	1
5	2	1
6	2	1
7	2	1
8	2	1
9	2	1
10	2	1
11	2	1
12	2	1
13	2	1
14	2	1
15	2	1
16	2	1
17	2	1
18	2	1
19	2	1
20	2	1
21	2	1

for accuracy of eigenvalues are Intel/AMD 32-bit processors, which have the x87 co-processor. Unfortunately, this feature is being phased out in modern 64-bit architectures.

7.2.2. A Gauss–Laguerre tridiagonal matrix

We have analysed the tridiagonal matrix in Gauss–Laguerre quadrature [17,25] for $n = 10$ and $\alpha = -0.75$. The symbol α (without the subscript) defines the weighting factor in Gauss–Laguerre quadrature. The tridiagonal is defined by

$$\alpha_i = 2i - 1 + \alpha,$$
$$\beta_i = \sqrt{i(i+\alpha)}.$$

We note that $z_i = \beta_i^2$ and hence it is not necessary to take square-roots in this and subsequent examples, where β_i is expressed as a square-root. The eigenvalues of the above and similar tridiagonals provide the abscissae for quadrature rules and hence accurate determination is paramount. They were previously tabulated by Concus et al. [6] for various values of α and n. Golub and Welsch [17] computed them with the QR-algorithm for $n = 10$ and $\alpha = -0.75$. Table 6 indicates that the eigenvalues inclusions can be computed to the best possible accuracy in double extended precision. However, in double precision, some of the diameters are high as shown in Table 7. We have also computed the eigenvalues in double precision by reversing the matrix,

$$\alpha_i \leftarrow \alpha_{n-i+1},$$
$$\beta_i \leftarrow \beta_{n-i},$$

Table 6
Inertia near the eigenvalues of the tridiagonal matrix associated with Gauss–Laguerre quadrature with $n = 10$ and $\alpha = -0.75$ in double extended precision

i	τ	τ hex end	inertia$^-$ ν^-, π^-, ζ^-	inertia$^{\leftrightarrow}$ $\nu^{\leftrightarrow}, \pi^{\leftrightarrow}, \zeta^{\leftrightarrow}$	inertia$^+$ ν^+, π^+, ζ^+	Fail
1	0.027666558670797241	5A9B	(0, 10, 0)	(0, 10, 0)	(0, 10, 0)	
1	0.027666558670797245	5A9C	(1, 9, 0)	(1, 9, 0)	(1, 9, 0)	
2	0.454784422605948535	A8A0	(1, 9, 0)	(1, 9, 0)	(1, 9, 0)	
2	0.454784422605948591	A8A1	(2, 8, 0)	(2, 8, 0)	(2, 8, 0)	
3	1.382425761158598609	2F77	(2, 8, 0)	(2, 8, 0)	(2, 8, 0)	
3	1.382425761158598831	2F78	(3, 7, 0)	(3, 7, 0)	(3, 7, 0)	
4	2.833980012092697010	A584	(3, 7, 0)	(3, 7, 0)	(3, 7, 0)	
4	2.833980012092697454	A585	(4, 6, 0)	(4, 6, 0)	(4, 6, 0)	
5	4.850971448764913596	19A5	(4, 6, 0)	(4, 6, 0)	(4, 6, 0)	
5	4.850971448764914484	19A6	(5, 5, 0)	(5, 5, 0)	(5, 5, 0)	
6	7.500010942642823863	2C18	(5, 5, 0)	(5, 5, 0)	(5, 5, 0)	
6	7.500010942642824752	2C19	(6, 4, 0)	(6, 4, 0)	(6, 4, 0)	
7	10.888408023834402982	BED9	(6, 4, 0)	(6, 4, 0)	(6, 4, 0)	
7	10.888408023834404759	BEDA	(7, 3, 0)	(7, 3, 0)	(7, 3, 0)	
8	15.199478044237601182	8A84	(7, 3, 0)	(7, 3, 0)	(7, 3, 0)	
8	15.199478044237602958	8A85	(8, 2, 0)	(8, 2, 0)	(8, 2, 0)	
9	20.789214621070104982	0569	(8, 2, 0)	(8, 2, 0)	(8, 2, 0)	
9	20.789214621070108535	056A	(9, 1, 0)	(9, 1, 0)	(9, 1, 0)	
10	28.573060164922104320	FB16	(9, 1, 0)	(9, 1, 0)	(9, 1, 0)	
10	28.573060164922107873	FB17	(10, 0, 0)	(10, 0, 0)	(10, 0, 0)	

Table 7
Interval diameters of the eigenvalues of the tridiagonal matrix associated with Gauss–Laguerre quadrature with $n = 10$ and $\alpha = -0.75$ in double extended precision

Eigenvalue	All machines double	All machines double reversed	Intel/AMD 32-bit double ext
1	64	74	1
2	27	18	1
3	7	7	1
4	4	4	1
5	3	3	1
6	3	3	1
7	2	2	1
8	2	2	1
9	2	1	1
10	3	1	1

and this operation does not alter the eigenvalues. The results are also recorded in the same table. It can be seen that by combining the results for the standard matrix and the reversed version a more optimal results can be generated.

7.3. Singular values

7.3.1. A Kac matrix

To evaluate interval singular values, we consider the real symmetric $n \times n$ Kac matrix T_n for even n defined by

$$\alpha_i = 0, \quad i = 1, n,$$
$$\beta_i = \sqrt{i(n-i)}, \quad i = 1, n-1.$$

Table 8
Inertia near the positive eigenvalues of Kac_{30} in double extended precision

i	τ	τ hex end	inertia$^-$ ν^-, π^-, ζ^-	inertia$^\leftrightarrow$ $\nu^\leftrightarrow, \pi^\leftrightarrow, \zeta^\leftrightarrow$	inertia$^+$ ν^+, π^+, ζ^+	Fail
16	0.999999999999999889	FFFF	(15, 15, 0)	(15, 15, 0)	(15, 15, 0)	
16	1.000000000000000000	0000	(15, 14, 1)	(15, 14, 1)	(15, 14, 1)	2
16	1.000000000000000222	0001	(16, 14, 0)	(16, 14, 0)	(16, 14, 0)	
17	2.999999999999999556	FFFF	(16, 14, 0)	(16, 14, 0)	(16, 14, 0)	
17	3.000000000000000000	0000	(17, 13, 0)	(17, 13, 0)	(16, 14, 0)	2
17	3.000000000000000444	0001	(17, 13, 0)	(17, 13, 0)	(17, 13, 0)	
18	4.999999999999999112	FFFF	(17, 13, 0)	(17, 13, 0)	(17, 13, 0)	
18	5.000000000000000000	0000	(18, 12, 0)	(17, 12, 1)	(17, 13, 0)	2
18	5.000000000000000888	0001	(18, 12, 0)	(18, 12, 0)	(18, 12, 0)	
19	6.999999999999999112	FFFF	(18, 12, 0)	(18, 12, 0)	(18, 12, 0)	
19	7.000000000000000000	0000	(19, 11, 0)	(19, 11, 0)	(18, 12, 0)	2
19	7.000000000000000888	0001	(19, 11, 0)	(19, 11, 0)	(19, 11, 0)	
20	8.999999999999998224	FFFF	(19, 11, 0)	(19, 11, 0)	(19, 11, 0)	
20	9.000000000000000000	0000	(20, 10, 0)	(20, 10, 0)	(19, 11, 0)	2
20	9.000000000000001776	0001	(20, 10, 0)	(20, 10, 0)	(20, 10, 0)	
21	10.999999999999998224	FFFF	(20, 10, 0)	(20, 10, 0)	(20, 10, 0)	
21	11.000000000000000000	0000	(21, 9, 0)	(20, 10, 0)	(20, 10, 0)	2
21	11.000000000000001776	0001	(21, 9, 0)	(21, 9, 0)	(21, 9, 0)	
22	12.999999999999998224	FFFF	(21, 9, 0)	(21, 9, 0)	(21, 9, 0)	
22	13.000000000000000000	0000	(22, 8, 0)	(22, 8, 0)	(21, 9, 0)	2
22	13.000000000000001776	0001	(22, 8, 0)	(22, 8, 0)	(22, 8, 0)	
23	14.999999999999998224	FFFF	(22, 8, 0)	(22, 8, 0)	(22, 8, 0)	
23	15.000000000000000000	0000	(23, 7, 0)	(23, 7, 0)	(22, 8, 0)	2
23	15.000000000000001776	0001	(23, 7, 0)	(23, 7, 0)	(23, 7, 0)	
24	16.999999999999996447	FFFF	(23, 7, 0)	(23, 7, 0)	(23, 7, 0)	
24	17.000000000000000000	0000	(24, 6, 0)	(24, 6, 0)	(23, 7, 0)	2
24	17.000000000000003553	0001	(24, 6, 0)	(24, 6, 0)	(24, 6, 0)	
25	18.999999999999996447	FFFF	(24, 6, 0)	(24, 6, 0)	(24, 6, 0)	
25	19.000000000000000000	0000	(25, 5, 0)	(24, 6, 0)	(24, 6, 0)	2
25	19.000000000000003553	0001	(25, 5, 0)	(25, 5, 0)	(25, 5, 0)	
26	20.999999999999996447	FFFF	(25, 5, 0)	(25, 5, 0)	(25, 5, 0)	
26	21.000000000000000000	0000	(26, 4, 0)	(25, 5, 0)	(25, 5, 0)	2
26	21.000000000000003553	0001	(26, 4, 0)	(26, 4, 0)	(26, 4, 0)	
27	22.999999999999996447	FFFF	(26, 4, 0)	(26, 4, 0)	(26, 4, 0)	
27	23.000000000000000000	0000	(27, 3, 0)	(26, 4, 0)	(26, 4, 0)	2
27	23.000000000000003553	0001	(27, 3, 0)	(27, 3, 0)	(27, 3, 0)	
28	24.999999999999996447	FFFF	(27, 3, 0)	(27, 3, 0)	(27, 3, 0)	
28	25.000000000000000000	0000	(28, 2, 0)	(28, 2, 0)	(27, 3, 0)	2
28	25.000000000000003553	0001	(28, 2, 0)	(28, 2, 0)	(28, 2, 0)	
29	26.999999999999996447	FFFF	(28, 2, 0)	(28, 2, 0)	(28, 2, 0)	
29	27.000000000000000000	0000	(29, 1, 0)	(29, 1, 0)	(28, 2, 0)	2
29	27.000000000000003553	0001	(29, 1, 0)	(29, 1, 0)	(29, 1, 0)	
30	28.999999999999996447	FFFF	(29, 1, 0)	(29, 1, 0)	(29, 1, 0)	
30	29.000000000000000000	0000	(29, 0, 1)	(29, 0, 1)	(29, 0, 1)	2
30	29.000000000000003553	0001	(30, 0, 0)	(30, 0, 0)	(30, 0, 0)	

It is well known that the eigenvalues of this matrix are integers and they are given by $-n, -n+1, \ldots, n-1, n$. This matrix can be considered as the Golub–Kahan form of the bidiagonal $(n/2) \times (n/2)$ matrix B,

$$a_i = \sqrt{(2i-1)(n-2i+1)}, \quad b_i = \sqrt{2i(n-2i)}.$$

The singular values of B are integers equal to $1, 2, \ldots, n-1, n$.

Half of the interval eigenvalues of the Kac matrix for $n = 30$ on AMD and Intel 32-bit machines in double extended precision are illustrated in Table 8; the other half is symmetric except for the signs. It can be seen that all interval diameters are equal to two and the open intervals straddles the exact eigenvalue given by an integer. In double precision, the seventeenth and the eighteenth eigenvalues were computed with less accuracy compared with double extended results; the details are illustrated in Table 9. The interval diameters for this example are compared in Table 10.

Table 9
Inertia near the 17th and 18th eigenvalues of Kac_{30} in double precision

i	τ	τ hex end	inertia$^-$ ν^-, π^-, ζ^-	inertia$^\leftrightarrow$ $\nu^\leftrightarrow, \pi^\leftrightarrow, \zeta^\leftrightarrow$	inertia$^+$ ν^+, π^+, ζ^+	Fail
17	2.999999999999999112	FFFE	(16, 14, 0)	(16, 14, 0)	(16, 14, 0)	
17	2.999999999999999556	FFFF	(17, 13, 0)	(16, 14, 0)	(16, 14, 0)	2
17	3.000000000000000000	0000	(17, 13, 0)	(17, 13, 0)	(16, 14, 0)	2
17	3.000000000000000444	0001	(17, 13, 0)	(17, 13, 0)	(16, 14, 0)	2
17	3.000000000000000888	0002	(17, 13, 0)	(17, 13, 0)	(17, 13, 0)	
18	4.999999999999999112	FFFF	(17, 13, 0)	(17, 13, 0)	(17, 13, 0)	
18	5.000000000000000000	0000	(18, 12, 0)	(17, 12, 1)	(17, 13, 0)	2
18	5.000000000000000888	0001	(18, 12, 0)	(18, 12, 0)	(17, 12, 1)	2
18	5.000000000000001776	0002	(18, 12, 0)	(18, 12, 0)	(18, 12, 0)	

Table 10
Interval diameters of the eigenvalues of Kac_{30}

Eigenvalue	All machines double	Intel/AMD 32-bit double extended
16	2	2
17	4	2
18	3	2
19	2	2
20	2	2
21	2	2
22	2	2
23	2	2
24	2	2
25	2	2
26	2	2
27	2	2
28	2	2
29	2	2
30	2	2

Table 11
Inertia near the positive eigenvalues of the Golub–Kahan form of the Gauss–Laguerre bidiagonal matrix in double extended precision

i	τ	τ hex end	inertia$^-$ ν^-, π^-, ζ^-	inertia$^\leftrightarrow$ $\nu^\leftrightarrow, \pi^\leftrightarrow, \zeta^\leftrightarrow$	inertia$^+$ ν^+, π^+, ζ^+	Fail
1	−5.345377457665839493	CD37	(0, 20, 0)	(0, 20, 0)	(0, 20, 0)	
1	−5.345377457665838605	CD36	(1, 19, 0)	(1, 19, 0)	(1, 19, 0)	
2	−4.559519121691465671	9B20	(1, 19, 0)	(1, 19, 0)	(1, 19, 0)	
2	−4.559519121691464782	9B1F	(2, 18, 0)	(2, 18, 0)	(2, 18, 0)	
3	−3.898650797934793744	3BE4	(2, 18, 0)	(2, 18, 0)	(2, 18, 0)	
3	−3.898650797934793300	3BE3	(3, 17, 0)	(3, 17, 0)	(3, 17, 0)	
4	−3.299758782674031554	309D	(3, 17, 0)	(3, 17, 0)	(3, 17, 0)	
4	−3.299758782674031110	309C	(4, 16, 0)	(4, 16, 0)	(4, 16, 0)	
5	−2.738614785369206661	05BD	(4, 16, 0)	(4, 16, 0)	(4, 16, 0)	
5	−2.738614785369206217	05BC	(5, 15, 0)	(5, 15, 0)	(5, 15, 0)	
6	−2.202492099591940811	90EB	(5, 15, 0)	(5, 15, 0)	(5, 15, 0)	
6	−2.202492099591940367	90EA	(6, 14, 0)	(6, 14, 0)	(6, 14, 0)	
7	−1.683442904316240218	8B4D	(6, 14, 0)	(6, 14, 0)	(6, 14, 0)	
7	−1.683442904316239996	8B4C	(7, 13, 0)	(7, 13, 0)	(7, 13, 0)	
8	−1.175766031640053333	4487	(7, 13, 0)	(7, 13, 0)	(7, 13, 0)	
8	−1.175766031640053111	4486	(8, 12, 0)	(8, 12, 0)	(8, 12, 0)	
9	−0.674377062633322866	D865	(8, 12, 0)	(8, 12, 0)	(8, 12, 0)	
9	−0.674377062633322755	D864	(9, 11, 0)	(9, 11, 0)	(9, 11, 0)	
10	−0.166332674693811272	76D3	(9, 11, 0)	(9, 11, 0)	(9, 11, 0)	
10	−0.166332674693811244	76D2	(10, 10, 0)	(10, 10, 0)	(10, 10, 0)	
11	0.166332674693811244	76D2	(10, 10, 0)	(10, 10, 0)	(10, 10, 0)	
11	0.166332674693811272	76D3	(11, 9, 0)	(11, 9, 0)	(11, 9, 0)	
12	0.674377062633322755	D864	(11, 9, 0)	(11, 9, 0)	(11, 9, 0)	
12	0.674377062633322866	D865	(12, 8, 0)	(12, 8, 0)	(12, 8, 0)	
13	1.175766031640053111	4486	(12, 8, 0)	(12, 8, 0)	(12, 8, 0)	
13	1.175766031640053333	4487	(13, 7, 0)	(13, 7, 0)	(13, 7, 0)	
14	1.683442904316239996	8B4C	(13, 7, 0)	(13, 7, 0)	(13, 7, 0)	
14	1.683442904316240218	8B4D	(14, 6, 0)	(14, 6, 0)	(14, 6, 0)	
15	2.202492099591940367	90EA	(14, 6, 0)	(14, 6, 0)	(14, 6, 0)	
15	2.202492099591940811	90EB	(15, 5, 0)	(15, 5, 0)	(15, 5, 0)	
16	2.738614785369206217	05BC	(15, 5, 0)	(15, 5, 0)	(15, 5, 0)	
16	2.738614785369206661	05BD	(16, 4, 0)	(16, 4, 0)	(16, 4, 0)	
17	3.299758782674031110	309C	(16, 4, 0)	(16, 4, 0)	(16, 4, 0)	
17	3.299758782674031554	309D	(17, 3, 0)	(17, 3, 0)	(17, 3, 0)	
18	3.898650797934793300	3BE3	(17, 3, 0)	(17, 3, 0)	(17, 3, 0)	
18	3.898650797934793744	3BE4	(18, 2, 0)	(18, 2, 0)	(18, 2, 0)	
19	4.559519121691464782	9B1F	(18, 2, 0)	(18, 2, 0)	(18, 2, 0)	
19	4.559519121691465671	9B20	(19, 1, 0)	(19, 1, 0)	(19, 1, 0)	
20	5.345377457665838605	CD36	(19, 1, 0)	(19, 1, 0)	(19, 1, 0)	
20	5.345377457665839493	CD37	(20, 0, 0)	(20, 0, 0)	(20, 0, 0)	

7.3.2. The bidiagonal Cholesky factor of the Gauss–Laguerre tridiagonal

It is well known that singular values of bidiagonal matrices can be computed to high accuracy [2,7]. Hence we have used the bidiagonal Cholesky factor of the Gauss–Laguerre tridiagonal in our analysis. The bidiagonal elements of this Cholesky factor are given by [25],

$$a_i = \sqrt{2i - 1 + \alpha},$$
$$b_i = \sqrt{i(i + \alpha)}.$$

Table 12
Interval diameters of the eigenvalues of the Golub–Kahan form of the Gauss–Laguerre bidiagonal matrix

Eigenvalue	All machines double	Intel/AMD 32-bit double extended
11	9	1
12	4	1
13	2	1
14	2	1
15	2	1
16	2	1
17	2	1
18	2	1
19	2	1
20	2	1

The Golub–Kahan form is a 20×20 matrix and the 11th–20th eigenvalues of the Golub–Kahan form are the singular values of the bidiagonal matrix. Thus, the abscissae are given by the squares of the singular values. Table 11 indicates that the eigenvalue inclusions can be computed to the highest accuracy possible. The corresponding Table 12 shows that the interval diameters in double precision are not as small as in the double extended case. However, by comparing the Tables 7 and 12, it can be seen that diameters given by the Cholesky factor are superior to that given by the original tridiagonal. However, this comparison is slightly unfair since the machine numbers in the neighbourhood of the eigenvalues are different from the corresponding singular values. Furthermore, if abscissae are required for quadrature then the singular values have to be squared, which may incur one ulp errors.

8. Conclusions

We have developed new algorithms to compute accurate eigenvalues (singular values) of symmetric tridiagonal (bidiagonal) matrices. The algorithms provide unprecedented accuracy and hence useful in their own right and also to debug and validate other less accurate algorithms. Such accuracy is also required to compute accurate eigenvectors [9,10] since current eigenvector algorithms perform poorly as demonstrated in [12].

In our numerical experiments, we have restricted ourselves to matrices which can be represented exactly in floating-point arithmetic. In practice, there will be errors due to decimal to binary conversion but they should be less than one ulp for each element. It would be an interesting exercise to find and compare the inclusions of eigenvalues (singular values) with different rounding modes in the decimal to binary conversion and then to assess these results with the error analysis given in [11].

Acknowledgments

Author is grateful to his former colleague Michael Pont for extensive comments on a previous version of this report; this led to many improvements. Many discussions with Professor William Kahan are gratefully acknowledged. Author wishes to thank Timothy Fernando for computational

support, the Oxford Supercomputing Centre for access to 64-bit Intel Xeon and AMD Opteron based parallel computers and the two referees for careful reading of the manuscript.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Blackford, D. Sorensen, LAPACK Users' Guide, Release 3.0., SIAM, Philadelphia, 1999.
[2] J. Barlow, J. Demmel, Computing accurate eigensystems of scaled diagonally dominant matrices, SIAM J. Numer. Anal. 27 (1990) 762–791.
[3] J.L. Barlow, More accurate bidiagonal reduction for computing the singular value decomposition, SIAM J. Matrix Anal. Appl. 23 (2002) 761–798.
[4] L.S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, H. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, D. Sorensen, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.
[5] J.C. Commercon, Eigenvalues of tridiagonal symmetric interval matrices, IEEE Trans. Automat. Control 39 (1994) 377–379.
[6] P. Concus, D. Cassatt, G. Jaehnig, E. Melby, Tables for the evaluation of $\int_0^\infty x^\beta \mathrm{e}^{-x} f(x)\,\mathrm{d}x$ by Gauss–Laguerre quadrature, Math. Comp. 17 (1963) 245–256.
[7] J. Demmel, W. Kahan, Accurate singular values of bidiagonal matrices, SIAM J. Sci. Stat. Comput. 11 (1990) 873–912.
[8] J.W. Demmel, I. Dhillon, H. Ren, On the correctness of some bisection-like parallel algorithms in floating point arithmetic, Electron. Trans. Numer. Anal. 3 (1995) 116–149.
[9] K.V. Fernando, Computing an eigenvector of a tridiagonal when the eigenvalue is known, Z. Angew. Math. Mech. 76 (suppl. 1) (1996) 299–302.
[10] K.V. Fernando, On computing an eigenvector of a tridiagonal matrix. Part I: Basic results, SIAM J. Matrix Anal. Appl. 18 (1997) 1013–1034.
[11] K.V. Fernando, Accurately counting singular values of bidiagonal matrices and eigenvalues of skew-symmetric tridiagonal matrices, SIAM J. Matrix Anal. Appl. 20 (1998) 373–399.
[12] K.V. Fernando, Accurate ordering of eigenvectors and singular vectors without eigenvalues and singular values, Linear Algebra Appl. 374 (2003) 1–17.
[13] K.V. Fernando, B.N. Parlett, Accurate singular values and differential qd algorithms, Numer. Math. 67 (1994) 191–229.
[14] F.R. Gantmacher, Theory of Matrices, vol. 2, Chelsea, New York, 1989.
[15] G.H. Golub, W. Kahan, Calculating the singular values and pseudo-inverse of a matrix, SIAM J. Numer. Anal. 2 (1965) 205–224.
[16] G.H. Golub, C.F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, 1996.
[17] G.H. Golub, J.H. Welsch, Calculation of Gauss quadrature rules, Math. Comp. 23 (1969) 221–239.
[18] B. Hayes, A lucid interval, Amer. Scient. 91 (2003) 484–488.
[19] T. Hickey, Q. Ju, M.H. van Emden, Interval arithmetic: from principles to implementation, J. ACM 48 (2001) 1038–1068.
[20] R.A. Horn, C.R. Johnson, Matrix Analysis, Cambridge University Press, Cambridge, 1985.
[21] IEEE, Standard for Binary Floating Point Arithmetic 754-1985, ANSI/IEEE, New York, 1985.
[22] ISO/IEC, 9899:1999, Programming Languages – C, ISO, 1999.
[23] C. Jordan, Mèmoire sur les formes bilinèaires, J. Math. Pures Appl. 19 (1874) 35–54.
[24] W. Kahan, Accurate eigenvalues of a symmetric tri-diagonal matrix, Technical Report CS 41, Computer Science Department, Stanford University, 1966.
[25] D.P. Laurie, Questions related to Gaussian quadrature formulas and two-term recursions, in: W. Gautschi, G.H. Golub, G. Opfer (Eds.), Applications and Computation of Orthogonal Polynomials, Birkhäuser, Basel, 1999, pp. 133–144.
[26] R. Pavec, Some algorithms providing rigorous bounds for the eigenvalues of a matrix, J. Univ. Comput. Sci. 1 (1995) 548–559.
[27] G.W. Stewart, J.-G. Sun, Matrix Perturbation Theory, Academic Press, San Diego, 1990.
[28] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.