`Refineig`: a Program to Refine Eigensystems

§0: Abstract

Software to compute eigenvalues and eigenvectors of matrices can hardly be deemed infallible. Results are often rather less accurate than deserved by the data, sometimes far less accurate than different software could have provided at the same cost. `Refineig` attempts to tidy up those results at less cost than if the whole computation were repeated carrying higher precision throughout. `Refineig` uses a novel iterative refinement algorithm designed to cope well with the most common cause of inaccuracy, namely isolated pairs of *nearly* coincident eigenvalues. (No way can exist to cope economically with all possible pathologies.) Examples abound for which `refineig` improves accuracy spectacularly, but usually the improvement is modest. Curiously, `refineig` works far better with the more popular floating-point arithmetics than with the more expensive ones (though all purport to conform to the same IEEE Standard 754 for floating-point arithmetic), and works far better with earlier than latest versions of MATLAB.

Contents

§1: Introduction to Troubles

Function `eig` in MATLAB [1993] is typical of good software for computing eigensystems. The MATLAB assignment `[Q, V] = eig(B)` is intended to deliver a nonsingular matrix Q whose columns are the eigenvectors of the given square matrix B , and a diagonal matrix V of its eigenvalues. Ideally we should find $B = Q \cdot V \cdot Q^{-1}$ exactly. Instead, rounded arithmetic can at best produce the eigensystem of some nearby matrix $B - \Delta B := Q \cdot V \cdot Q^{-1}$ perturbed by a ΔB comparable with roundoff in B . Perturbations ΔB rather worse than several rounding errors can occur; the probability of encountering them increases with dimension.

Even when ΔB is minuscule, Q and V can be grossly inaccurate for systematic reasons that are artifacts of the algorithms `eig` uses. For example, although B and its transpose `B'` have the same eigenvalues with the same sensitivities to perturbation, examples will be exhibited for which `eig(B)` is systematically and substantially less accurate than `eig(B.')`. (Here `B.'` is MATLAB's notation for B^T .) Such examples are symptoms of `eig`'s susceptibility to mathematically trivial but numerically damaging accidents in the presentation of data.

All eigensystem-computing programs, not just `eig` , suffer from comparable susceptibilities, if recent work by Ming Gu [1995] can be taken at face value. He showed why reasonable attempts to attenuate the susceptibility of computed eigensystems to roundoff must overcome combinatorial obstacles which, in worst cases, grow exponentially more complex as dimension increases. Those obstacles frustrate attempts by software to decide reliably whether a computed invariant subspace of B should be left unreduced or reduced further to a sum of ultimately ***I**rreducible **I**nvariant **S**ubspaces*. Here are some examples to illustrate the difficulty:

Consider $B_j(x) = C \cdot X_j(x) \cdot C^{-1}$ where C is any well-conditioned 6-by-6 matrix and $X_j(x)$ is one of

$$X_1(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ x^2 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & 0 \\ 0 & -x^2 & 0 & 0 & 0 & 1 \\ x^4 & 0 & 0 & 0 & -x^2 & 0 \end{bmatrix}, \quad X_2(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ x^3 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & 0 \\ 0 & -x^4 & 0 & 0 & 0 & 1 \\ x^4 & 0 & 0 & 0 & -x^3 & 0 \end{bmatrix}, \quad X_3(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ x^3 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -x^3 & 0 & 0 \end{bmatrix}$$

$$X_4(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ x^4 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad X_5(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ x^5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{or} \quad X_6(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ x^6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

If $x \neq 0$ all have the same six distinct eigenvalues of magnitude $|x|$ since each $B_j(x)^6 = x^6 I$, but the noise from roundoff makes their numerically Irreducible Invariant Subspaces difficult to discern both unambiguously and robustly, especially as decreasing $|x|$ carries first x^6, then x^5, ..., then x^2 below the noise level. The number of discernable invariant subspaces for each $B_j(x)$ should drop as x decreases through various thresholds, different for each j, until $|x|$ becomes so tiny that the numerically **I**rreducible **I**nvariant **S**ubspaces become indistinguishable from those of $B_j(0)$. The respective numbers $\#IIS_j$ of those final subspaces are ...

$$\#IIS_1 = \#IIS_2 = 3, \qquad \#IIS_4 = \#IIS_5 = 2, \qquad \#IIS_3 = \#IIS_6 = 1.$$

Since $B_j(0)$ has at most three eigenvectors, one per irreducible invariant subspace, `eig` should fail to diagonalize it; but `[Q,V] = eig(`$B_j(0)$`)` still delivers six alleged eigenvectors in Q and their eigenvalues in V, usually inconsistent with `eig(`$B_j(0)$`.')`. Instead of zeros, the computed eigenvalues are typically of the order of 10^{-3} or 10^{-4} when the elements of $B_j(0)$ are single-digit integers. The simplest way to tell that `eig`'s results are dubious is to observe how ill-conditioned Q is; typically `rcond(Q)` falls well below 10^{-9}.

In general matrices are called "Defective" when, like $B_j(0)$, they have too few eigenvectors to permit diagonalization by any similarity transformation. In the n^2 - dimensional space of all n-by-n matrices, the defective matrices are dense in the (n^2-1) - dimensional algebraic variety (a hypersurface) of degree $n(n-1)$ that contains all matrices with repeated eigenvalues; this explains (Demmel [1988]) why defective matrices are almost never encountered at random, but near–defective matrices occur too often to ignore. To distinguish reliably among different kinds of defective and near-defective matrices costs too much more than users of eigensystem-computing software like `eig` are inclined to pay at first, so `eig` makes no distinction and delivers inaccurate results impartially for all kinds. More discriminating software must work harder and longer starting from a Schur decomposition, which lies beyond the purview of this opus; see instead Demmel and Kågström [1993] and Edelman, Elmroth and Kågström [1997] and references cited therein.

§2: `Refineig` to the Rescue

`Refineig` starts from a nonsingular matrix Q of approximate eigenvectors and a diagonal matrix V of approximate eigenvalues of a square matrix B ; say `[Q, V] = eig(B)` in MATLAB. Then `[Q, V] = refineig(Q, V, B)` performs a step of iterative refinement that replaces the old [Q, V] by a new. This step may be repeated. At most a few repetitions suffice because `refineig` converges very quickly (cubically) if it converges at all.

`Refineig` is designed to enhance the accuracy of a computed eigensystem when the given data are nearly defective or, failing that, to let the ever-worsening ill-condition of successively "improved" eigenvector matrices Q signal intractable closeness to defective data. Sometimes a nearly defective matrix looks that way only because it is badly scaled; then multiplying and dividing its rows and columns by suitably chosen scale factors may reduce its eigenproblem's ill-condition by orders of magnitude. This is what MATLAB's `balance` was intended to achieve, but it is fallible, so MATLAB offers a way to inhibit it: invoke `eig(..., 'nobalance')`. Both `eig` and `Refineig` may get better results after `nobalance`. `Refineig` also succeeds quite often when the data consist of small integers or other numbers stored exactly, or when the data include many zeros in locations that `eig` cannot exploit, or when data include repetitions and correlations that `eig` cannot protect from roundoff.

Hermitian and real symmetric matrices constitute another family of special cases. Though never defective, they can be so badly scaled (and ineligible for `balance`) that `eig` computes the smaller eigenvalues and their eigenvectors with poor relative accuracy. In these cases too, `refineig` usually recovers the accuracy the data deserve, but at the cost of invoking a peculiar algorithm required to preserve orthogonality among eigenvectors.

`Refineig` is not infallible. Worse, there is no easy way to tell whether it worked nor how well. Sometimes the improvement in an approximate eigensystem [Q, V] is apparent from a smaller residual $\Delta R := B \cdot Q - Q \cdot V$. More often the first few successively refined pairs [Q, V] converge convincingly. Since `refineig` can diverge or divagate, prudence requires a check that it has not much exacerbated the residual.

`Refineig` works better on some computers than on others. It works best on Intel-based IBM PCs and their clones, including those with Cyrix or AMD processors, and on Motorola 680x0-based Apple Macintoshes (not Power–Macs). It works less well on Apple Power–Macs and on the rarer and more expensive IBM Power–PCs and RS/6000s, Sun SPARCs, Silicon Graphics MIPS, Hewlett-Packard PA RISCs and DEC Alphas. `Refineig` could be reprogrammed to run more accurately (and slower) on Power–Macs and Power–PCs, but not in MATLAB. Why the difference? It all comes down to the accuracy of residuals.

§3: Residuals

The effectiveness of iterative refinement is generally limited by the accuracy to which residuals like $\Delta R := B \cdot Q - Q \cdot V$ can be computed. If Q and V were perfectly correct ΔR should vanish, so it must incur massive cancellation. Cancellation is not what limits accuracy; that is limited by the precision to which scalar products are accumulated during matrix multiplications.

When $B \cdot Q$ and $Q \cdot V$ are computed using arithmetic no more precise than the 53 sig. bit 8-byte wide formats in which B, Q and V are stored in memory, $\Delta R = B \cdot Q - Q \cdot V$ rarely rises much above the roundoff that accumulated during its computation. Only in those rare cases can `refineig` improve Q and V substantially. Those rare cases do occur often enough to justify `refineig`'s existence, as we shall see. Still, `refineig` pays off more often and better when ΔR is accumulated extra-precisely, thereby standing well above the roundoff noise generated during its computation. This is feasible on the computers most widely in use to-day:

Intel's Itanium, x86/87 and Pentiums, Cyrix and AMD clones, and Motorola 68040 or earlier 680x0 + 68881/2 processors can perform their floating-point arithmetic in registers 10 bytes wide with 64 instead of 53 sig. bits. These wider formats accommodate the narrower operands loaded from memory. Although MATLAB stores every subexpression in 8-byte memory cells, it can accumulate scalar products in those computers' 10-byte registers during matrix multiplication, sometimes doing so faster than if every partial sum were rounded to 8 bytes. To test for extra-precise accumulation evaluate MATLAB expressions like

```
[ (2 - 2^33) ,   2^33 ,   -1 ]*[ (1 + 2^32) ;   2^32 ;   1 ]  ,
```

which yields +1 correctly when this scalar product is accumulated from left to right with 64 sig. bits but yields the wrong sign when accumulated with only 53 sig. bits. However, see "MATLAB's Loss is Nobody's Gain", `<www.cs.berkeley.edu/~wkahan/MxMulEps.pdf>`.

To exploit extra-precise registers on machines that have them, `refineig` computes each of its residuals by performing one MATLAB matrix multiplication; for example $\Delta R := B \cdot Q - Q \cdot V$ is obtained from `dR = [B, Q]*[Q; -V]`. Another residual $\Delta^2 R := Q \cdot G - \Delta R$ is obtained from `d2R = [Q, dR]*[G; -I]` for an aptly dimensioned identity matrix I. This $\Delta^2 R$ figures in the computation of $\Delta C = Q^{-1} \Delta R$ which starts with a first approximation $G \approx Q^{-1} \Delta R$ and then refines it iteratively into a better approximation $\Delta C := G - Q^{-1} \Delta^2 R$ in order to attenuate the damage caused by a poor approximation to Q^{-1} when Q is extremely ill-conditioned. The cost of computing a residual in one MATLAB matrix multiplication is a near doubling of temporary memory residency and the time for arithmetic; neither cost need be incurred when `refineig` is programmed in Fortran or ***C*** with compiler support for register–wide floating–point variables.

MATLAB matrix products like `[B, Q]*[Q; -V]` produce the same residuals as `B*Q - Q*V` on machines that lack extra-precise registers. Extra-precise accumulation on such machines is awkward to program in MATLAB and runs slowly no matter how it is programmed. It is not too slow to be worth programming on Power-Macs/PCs and Itaniums; these machines have a "fused" multiply-accumulate instruction which needs to be invoked only twice to deliver the exact 16-byte product of two 8-byte floating-point numbers. Other machines require over a dozen instructions to do that. In any event, 16-byte accumulated matrix multiplication is easier to program not in MATLAB's language but in a language like Fortran provided its compiler supports REAL*16 ; in that event the eigensystem might be better computed using REAL*16 throughout without recourse to `refineig`. This brings us to the brink of a recurring question:

" When is high-precision arithmetic more economical than clever mathematics? "

The short answer is "Surprisingly often." The long answer lies beyond the purview of this opus.

§4: How `Refineig` Works for Non-Hermitian B

Assumed given are a diagonalizable non-Hermitian square matrix B and an approximation [Q, V] to its eigensystem good enough that the residual $\Delta R := B\cdot Q - Q\cdot V$ is small and Q is not too ill-conditioned. Of course V is diagonal. We wish to compute small corrections ΔQ and ΔV, the latter diagonal, for which ideally $B\cdot(Q+\Delta Q) = (Q+\Delta Q)\cdot(V+\Delta V)$ exactly. But ΔQ is not determined uniquely by this equation because it allows $Q+\Delta Q$ to be postmultiplied by any diagonal matrix. Our first task is to remedy this non-uniqueness.

Let $\Delta Z := Q^{-1}\Delta Q$ so $Q+\Delta Q = Q\cdot(I+\Delta Z)$. Postmultiplying this by $\mathrm{Diag}((I+\Delta Z))^{-1}$ defines a new ΔZ with zeros on its diagonal, which makes sense if every column of Q is already so nearly an eigenvector that it can be corrected by adding a tiny linear combination of other columns. Thus we fix $\Delta Q = Q\cdot\Delta Z$ uniquely by imposing the condition $\mathrm{diag}(\Delta Z) = o$ upon what shall be computed. (Notation: diag(...) is a vector, Diag(...) is a diagonal matrix.)

Next define $\Delta C := Q^{-1}\Delta R = Q^{-1}B\cdot Q - V$, to be computed as accurately as possible with the aid of iterative refinement: new $\Delta C := \Delta C - Q^{-1}(Q\cdot\Delta C - \Delta R)$. Whether the last residual term is accumulated extra-precisely or not, refinement is necessary to provide ΔC with a degree of protection against inaccuracy in cases when eigenvalues or eigenvectors' elements span a very wide range of magnitudes. Now that ΔC is as accurate we can make it, what good is it?

The equation $B\cdot(Q+\Delta Q) = (Q+\Delta Q)\cdot(V+\Delta V)$ that we wish to solve for ΔQ and diagonal ΔV is transformed by the substitutions $B = Q\cdot(V+\Delta C)\cdot Q^{-1}$ and $\Delta Q = Q\cdot\Delta Z$ into a new equation $(V+\Delta C)\cdot(I+\Delta Z) = (I+\Delta Z)\cdot(V+\Delta V)$, which simplifies by cancellation into a form

$$\Delta Z\cdot(V + \Delta V) - V\cdot\Delta Z = \Delta C\cdot(I + \Delta Z) - \Delta V \qquad (0)$$

that can be solved iteratively for ΔV and ΔZ. The left-hand side's diagonal vanishes because V and ΔV are diagonal and $\mathrm{diag}(\Delta Z) = o$; therefore, from any estimate of ΔZ,

$$\Delta V := \mathrm{diag}(\Delta C + \Delta C\cdot\Delta Z) = \mathrm{diag}(\Delta C) + O(\Delta...)^2 \qquad (1)$$

can be calculated. With an estimate of ΔV in hand we proceed to re–estimate ΔZ as follows.

First let u be the column vector whose elements are all 1's, and condense diagonal matrices V and ΔV into column vectors $v = \mathrm{diag}(V)$ and $\Delta v = \mathrm{diag}(\Delta V)$, and then compute

$$E := (u\cdot v^T - v\cdot u^T) + (u\cdot\Delta v^T - \Delta V) + \infty I \ . \qquad (2)$$

Its elements will all be nonzero if the eigenvalues of B are sufficiently different; otherwise, if division by zero occurs in the next step, the consequences will have to be cleaned up later. The next step entails *elementwise* division by E to produce

$$\Delta Z := ((\Delta C + \Delta C\cdot\Delta Z) - \Delta V)/E = (\Delta C - \Delta V)/E + O(\Delta...)^2 \ . \qquad (3)$$

Note that $\mathrm{diag}(\Delta Z) = o$ automatically because of the way ΔV was computed. If overflow or division by zero creates an ∞ or NaN in ΔZ, replace it by 0 to confine the damage to those eigenvectors of B belonging to almost coincident eigenvalues.

Formulas (1), (2) and (3) may be used iteratively to refine guesses $\{\Delta Z, \Delta V\}$. Starting with $\Delta Z := O$ in their right-hand sides, one pass through these formulas produces a new ΔZ in error by terms of order $O(\Delta...)^2$; a second pass produces ΔZ in error by $O(\Delta...)^3$. After ΔZ is refined enough, $\Delta Q := Q\cdot\Delta Z$ can be computed, and then $Q+\Delta Q$ and $V+\Delta V$.

All that would be fine if it actually worked; but it fails to converge in situations much like those that cause eig(B) to deliver inaccurate results, which is when refinement is most needed. The likeliest failure mode arises from pairs of nearly coincident eigenvalues. To combat that failure mode, `refineig` computes its starting ΔZ from a formula that would be perfect if ΔC were a permuted diagonal sum of 1-by-1 and 2-by-2 matrices, and is otherwise still correct in first order terms $O(\Delta...)$, unless eigenvalues collide. Here are some details:

Recall that, given ΔC and diagonal V, we wish to solve $(V+\Delta C)\cdot(I+\Delta Z) = (I+\Delta Z)\cdot(V+\Delta V)$ for ΔZ and diagonal ΔV subject to the constraint $\mathrm{diag}(\Delta Z) = o$. Further to simplify notation, let us temporarily rewrite $\Delta V = W$ and drop the Δ from ΔC and ΔZ. Then square C and diagonal V are given and we seek diagonal W and square Z with $\mathrm{diag}(Z) = o$ to satisfy
$(V + C)\cdot(I + Z) = (I + Z)\cdot(V + W)$ or, equivalently, $Z\cdot(V + W) - V\cdot Z = C\cdot(I + Z) - W$.
Apparently the columns of $I+Z$ are the eigenvectors of $V+C$ whose eigenvalues are on the diagonal of $V+W$. This eigenproblem can be solved easily in closed form when $V+C$ is a 2-by-2 matrix; our next task is to express that solution in a form that works also when $V+C$ is a permuted diagonal sum of 1-by-1 and 2-by-2 matrices.

Consider first the two eigenvalues $x = v_1 + w_1$ and $x = v_2 + w_2$ of

$$V + C = \begin{bmatrix} v_1 + c_1 & c_{12} \\ c_{21} & v_2 + c_2 \end{bmatrix} .$$

They are the two zeros x of a quadratic

$$x^2 - (v_1 + c_1 + v_2 + c_2)\cdot x + (v_1 + c_1)\cdot(v_2 + c_2) - c_{12}\cdot c_{21}$$

whose discriminant $y^2 = \frac{1}{4}\cdot(v_1 - v_2 + c_1 - c_2)^2 + c_{12}\cdot c_{21}$ must not vanish unless c_{12} and c_{21} vanish too lest $V+C$ be non-diagonalizable. Let

$$s := \tfrac{1}{2}\cdot((v_1 - v_2) + (c_1 - c_2))$$

and choose the sign of (presumably nonzero)

$$y := \pm\sqrt{(s^2 + c_{12}\cdot c_{21})}$$

so as to maximize the (consequently nonzero) magnitude of

$$h := s + y .$$

Then the matrix of eigenvectors $I+Z$ turn out to have

$$Z = \begin{bmatrix} 0 & z_{12} \\ z_{12} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -c_{12}/h \\ c_{12}/h & 0 \end{bmatrix} ,$$

except that if $h = 0$ let $Z := O$; and the eigenvalues $\mathrm{diag}(V+W)$ of $V+C$ turn out to have

$$w_1 = c_1 + c_{12}\cdot z_{21} , \quad w_2 = c_2 + c_{21}\cdot z_{12} .$$

Note that $\det(I+Z) = 2y/h$ is nonzero whenever y is nonzero, and then Z is determined as accurately as cancellation in y^2 permits, which means no more than half the sig. bits carried will be lost. Also $|z_{12}\cdot z_{21}| \le 1$, so Z never gets very big unless it is very badly scaled.

With the foregoing considerations in mind, consider next what to do in the slightly more general case when ΔC is a permuted diagonal sum of 1-by-1 and 2-by-2 matrices:

Column vector u whose elements are all 1 's will be used again along with $\Delta v := \mathrm{diag}(\Delta C)$ and $v := \mathrm{diag}(V)$. First construct $\Delta Ç := \Delta C - \mathrm{Diag}(\Delta v)$ with zeros on its diagonal, then the (complex) skew matrix $S := ((u\cdot v^T - v\cdot u^T) + (u\cdot \Delta v^T - \Delta v\cdot u^T))/2$ and then the symmetric $T := \sqrt{}(S\bullet S + \Delta Ç^T\bullet \Delta Ç)$ *elementwise*, also with zeros on their diagonals. A skew Y must be formed from T by reversing signs of some of its elements in order to maximize the magnitude of every element of S+Y . To this end compute skew $K := \mathrm{Re}\{ S\bullet \mathrm{conj}(T) \}$ *elementwise* and reverse the sign of every element of T for which the corresponding element of K is either negative or subdiagonal and zero. (Here conj(T) is the complex conjugate of T .) Having thus formed Y , compute skew $H := S+Y$ and then $\Delta Z := \Delta Ç/(H + \infty I)$ *elementwise*. The addition of ∞I prevents divisions by zero on the diagonal but they may occur elsewhere; if so, replace any resulting ∞ or NaN in ΔZ by 0 .

The foregoing prescription can be followed also when ΔC is an arbitrary square matrix, not a permuted diagonal sum of 1-by-1 and 2-by-2 matrices; then the prescription provides approximations $\Delta V = \mathrm{Diag}(\Delta v)$ and ΔZ that satisfy the original equation (0) to within errors of order $O(\Delta\ldots)^2$. For this reason, `refineig` follows that prescription to get initial guesses for ΔZ and ΔV whose errors may be reduced to $O(\Delta\ldots)^3$ by one pass through (1), (2) and (3) as described previously.

That prescription is what most distinguishes `refineig` from previous attempts at iterative refinement like that of Dongarra, Moler and Wilkinson [1983]. Unlike theirs, `refineig`'s algorithm continues to work well when clustered eigenvalues consist of pairs of close but not coincident eigenvalues provided every such pair is separated well enough from all other eigenvalues; this is the most common failure mode for all previous attempts at refinement.

Of course, the foregoing computations must fail when Q is too nearly singular or $V + \Delta V$ has too many too nearly equal diagonal elements. For that reason prudence requires the new residual $B\cdot Q - Q\cdot V$ to be checked lest the "refinement" $V := V+\Delta V$ and $Q := Q+\Delta Q$ actually make matters worse. If smaller but not small enough, the new residual may be attenuated further by iterating refinement; say `[Q, V] = refineig( Q, V, B )` in MATLAB. If refinement converges it converges cubically (very fast) until roundoff gets in the way. To maintain compatibility with `eig` and protect repeated refinement from runaway scaling, `refineig` divides each column of the refined Q by its length, so the word "converges" can be taken literally.

If `refineig` will be invoked often enough, or for matrices B of dimensions large enough, to keep a computer busy for more than a few seconds, then it is worth reprogramming to exploit matrix operations faster than are built into MATLAB 3.5. For instance, multiplication by or subtraction of diagonal matrices should exploit their sparseness; and when two expressions $Q^{-1}\ldots$ are computed the second should re-use the first's triangular LU-factorization; these economies are realizable in MATLAB 4.2 . Memory occupancy could be reduced and time could be saved during the computation of residuals if MATLAB did not oblige us to construct and multiply double-sized matrices `[…, …]*[…; …]` in order to accumulate scalar products extra-precisely on hardware with this capability.

§5: How `Refineig` Works for Hermitian A

When A' = A is Hermitian we expect `eig(A)` to produce an orthogonal or unitary matrix Q of eigenvectors; it must satisfy Q'·Q = I . Actually roundoff and other artifacts in `eig` corrupt this equation. Neither need this equation be satisfied after Q is refined to a new Q := Q+ΔQ using the formulas of §4 , including the final normalization Q —> Q·√(Diag(Q' Q))$^{-1}$. Even if those formulas pull every column of Q closer to an eigenvector, they can actually push Q farther from unitary whenever some eigenvalues of A nearly coincide. For this reason alone, `refineig` must treat Hermitian matrix eigenproblems specially. Besides, Hermitian matrices cannot be defective, so they afford `refineig` no excuse for failure.

How does `refineig` refine an approximate eigensystem [Q, V] of a matrix A found to be Hermitian?

In the account that follows, symmetric letters A, H, I, M, O, T, U, V, W, X and Y are used to represent Hermitian matrices so as to distinguish them from others. That is why the input formerly called B has been renamed A . Letters O, S and Z will represent skew–Hermitian matrices; evidently O must be a zero matrix. The zero vector is o .

Suppose now that an eigenvector matrix Q and a real diagonal eigenvalue matrix V have been found approximately for a given Hermitian matrix A = A' , perhaps from `[Q, V] = eig(A)` .

If H is real symmetric MATLAB ensures that Q is nearly orthogonal but, since `refineig` knows nothing about the provenance of Q , it must be replaced by the nearest unitary matrix P := Q·(Q'·Q)$^{-1/2}$, which is generally best computed from a singular value decomposition of Q . There is a faster way, if residual ΔY := Q'·Q – I (obtained as `dY = [Q', I]*[[Q; -I]` in MATLAB) is small enough, because then P := Q – Q·ΔY/2 to working accuracy; and ΔY is surely small enough when 1 – ||ΔY||^2 rounds to 1 because then P'·P – I = ΔY^2·(ΔY – 3·I)/4 is negligible. One way or another, `refineig` obtains an accurately unitary P = Q – ΔQ .

The next task is to refine P towards the eigenvectors of A . The refinement must preseve orthogonality of the columns of P , so it has been put into the form P —> P – 2·P·(I+ΔZ)$^{-1}$ΔZ where ΔZ = –ΔZ' is a skew–Hermitian matrix whose Cayley Transform

$$I - 2\cdot(I+\Delta Z)^{-1}\Delta Z \;=\; (I+\Delta Z)^{-1}(I-\Delta Z)$$

is therefore unitary. It is a slightly special *proper* unitary matrix because it cannot have –1 as an eigenvalue. Because further postmultiplication by any unitary diagonal is allowed, the eigenvectors of A do not yet determine ΔZ uniquely; to determine ΔZ sharply we insist that

$$\mathrm{diag}(\Delta Z) = o\ .$$

Is this constraint always satisfiable? Yes. This is obviously so when A and therefore ΔZ are real, unobviously so when they are complex; for a proof see my note "Is there a Skew Cayley Transform with Zero Diagonal?" [1999].

The residual ΔH := P'·(A·P – P·V) = P·A·P – V , computed from `P'*([A, P]*[P; -V])` in MATLAB, would be Hermitian but for roundoff, whose effect is mitigated by replacing

ΔH by $(\Delta H+\Delta H')/2$. Then $(I+\Delta Z)^{-1}(I-\Delta Z)$ turns out to have to be a matrix of eigenvectors for $V+\Delta H$, whence follows $(V+\Delta H)\cdot((I+\Delta Z)^{-1}(I-\Delta Z)) = ((I+\Delta Z)^{-1}(I-\Delta Z))\cdot(V+\Delta V)$ wherein the eigenvalue correction ΔV is diagonal like V . Expanded and simplified, the last equation becomes

$$\Delta V - \Delta Z\cdot\Delta V\cdot\Delta Z - \Delta Z\cdot(2\cdot V+\Delta V) + (2\cdot V+\Delta V)\cdot\Delta Z = \Delta H + \Delta Z\cdot\Delta H - \Delta H\cdot\Delta Z - \Delta Z\cdot\Delta H\cdot\Delta Z \,. \quad (4)$$

It has to be solved iteratively for a real diagonal ΔV and a skew–Hermitian ΔZ . To that end it will be broken into its diagonal and off-diagonal parts.

First the diagonal: Recall that $\mathrm{diag}(\Delta Z) = o$ and define $|\Delta Z|^2 := -\Delta Z'\bullet\Delta Z$ *elementwise* to deduce how, from any such skew-Hermitian estimate of ΔZ , to compute an estimate of

$$\Delta v := (I + |\Delta Z|^2)^{-1}\,\mathrm{diag}(\,\Delta H + \Delta Z\cdot\Delta H - \Delta H\cdot\Delta Z - \Delta Z\cdot\Delta H\cdot\Delta Z\,) \quad (5)$$
$$= \mathrm{diag}(\Delta H) + O(\Delta\ldots)^2 \,. \qquad (\ \Delta v \text{ must be exactly real.})$$

Then $\Delta V := \mathrm{Diag}(\Delta v)$.

For the off-diagonal parts the column vector $u = [1, 1, ..., 1]^T$ and the vectors $v := \mathrm{diag}(V)$ and Δv will be needed again. They figure in the construction of a skew matrix

$$S := (2\cdot v+\Delta v)\cdot u^T - u\cdot(2\cdot v+\Delta v)^T = -S^T \quad (6)$$

whose off-diagonal entries are nonzero provided the eigenvalues of A are different enough that all entries of $2v+\Delta v$ are distinct; otherwise, if division by zero occurs in the next step, the consequences will have to be cleaned up later. The next step entails *elementwise* division by S to produce a new estimate of the skew–Hermitian

$$\Delta Z := (\,\Delta H - \Delta V + \Delta Z\cdot\Delta H - \Delta H\cdot\Delta Z - \Delta Z\cdot(\Delta H - \Delta V)\cdot\Delta Z\,)/(S + \imath\infty I) \quad (7)$$
$$= (\,\Delta H - \Delta V\,)/(S + \imath\infty I) + O(\Delta...)^2 \qquad (\ \imath = \sqrt{(-1)}\ )$$

which automatically sets $\mathrm{diag}(\Delta Z) = o$. The addition of $\imath\infty I$ to S prevents divisions by zero on the diagonal, but they may occur elsewhere; if so, replace any resulting ∞ or NaN in ΔZ by 0 to confine the damage to those eigenvectors of A belonging to eigenvalues that almost coincide. (Elements of ΔZ too much bigger than 1 in magnitude do more harm than good.)

As before, formulas (5), (6) and (7) may be used iteratively to refine guesses $\{\ \Delta Z, \Delta V\ \}$. Starting with $\Delta Z := O$ in their right-hand sides, one pass through these formulas normally produces a new ΔZ in error by terms of order $O(\Delta...)^2$; a second pass reduces ΔZ 's error to $O(\Delta...)^3$. After ΔZ is refined enough, $\Delta P := -2\cdot P\cdot(I+\Delta Z)^{-1}\Delta Z$ can be computed, and then $P+\Delta P$ and $V+\Delta V$. As before, the iteration can converge slowly or not at all if eigenvalues are too nearly coincident. To attack this defect, a first guess ΔZ better than O shall be derived the same way as was a formula attributed by Bodewig [1959] to Jacobi (1838), Jahn (1948) and Magnier (1948), though their formula would be valid only for real ΔH . Their formula has served successfully as a quadratically convergent iteration to compute eigenvalues of real symmetric matrices (no larger than 4-by-4) in a programmable shirt-pocket calculator, the HP-15C [1982]. The derivation begins, as before, by considering 2-by-2 Hermitian matrices ΔH , and extends these considerations to permuted diagonal sums ΔH of 1-by-1 and 2-by-2 matrices in such a way as provides for them exactly the skew solution ΔZ of (4) in a formula

that approximates ΔZ in general within an error $O(\Delta...)^2$. Here is the result of that derivation:

First $\mathrm{signum}(\mu) := \mu/|\mu|$ has to be defined at zero; $\mathrm{signum}(0) := 0$. Alas, $\mathrm{signum}(\infty)$ is problematic because so few complex infinities are representable as floating-point numbers, and those that are representable, like real $\pm\infty$, often have accidental signs. The algorithm that follows will treat division by zero as an extremely rare special case.

Next, for all finite μ define

$$ß(\mu) := \mathrm{signum}(\mu)\cdot\tan(\arctan(|\mu|)/2) = \mu/(1 + \sqrt{(1 + |\mu|^2)})$$ and
$$f(\mu) := \mathrm{signum}(\mu)\cdot\tan(\arctan(|\mu|)/4) = ß(ß(\mu)) .$$

On some computers $f(\mu)$ can be computed faster from tan and arctan than from two square roots and divisions. In any event, $|ß(\mu)| < 1$ and $|f(\mu)| < \sqrt{2} - 1 = 1/(1 + \sqrt{2})$ for all finite μ.

From vectors $u := [1, 1, ..., 1]^T$ and $y := (\mathrm{diag}(V) + \mathrm{diag}(\Delta H))/2$ compute skew–Hermitian

$$\Delta S := \Delta H/(y\cdot u^T - u\cdot y^T + \imath\infty I) \textit{ elementwise},$$ and
$$\Delta Z := f(\Delta S) \textit{ elementwise}$$

except that, wherever an element Δz_{ij} of ΔZ is found to be ∞ or NaN because of division by zero or overflow, it must be replaced by $\Delta z_{ij} := \mathrm{signum}((j–i)\cdot\Delta h_{ij})/(1 + \sqrt{2})$ using the corresponding element Δh_{ij} of ΔH. Thus, every element Δz_{ij} of ΔZ satisfies $\Delta z_{ij} = –\Delta z_{ij}$, $\mathrm{signum}(\Delta z_{ij}) = \pm\mathrm{signum}(\Delta h_{ij})$, and $|\Delta z_{ij}| \le 1/(1 + \sqrt{2}) = 0.41421356\ldots$.

The foregoing formula for ΔZ is perfect if ΔH is a permutation of a diagonal sum of 1-by-1 and 2-by-2 matrices, and correct to first order in ΔH otherwise, so it provides a first guess ΔZ better than O to start the iteration (5), (6) and (7). It also exposes one of `refineig`'s potential failure modes in equation (5) :

If A has a cluster of too many too nearly equal eigenvalues, ΔZ may have too many elements that are not very tiny, and then $(I + |\Delta Z|^2)$ in (5) may be singular or nearly so. Then iterative refinement of $(I + |\Delta Z|^2)^{-1}\ldots$ diminishes the risk of malfunction but does not eliminate it. The same palliative applied to $(I + \Delta Z)^{-1}\Delta Z$ could attenuate another hazard arising from the same cause when ΔZ computed in (7) has gargantuan elements, but a reasonable alternative is first to clip the magnitudes of excessively big elements of ΔZ .

§6: Tests

`Refineig` was implemented as a MATLAB .m-file function. While it was being debugged, its behavior was compared with the following expectations:

- Given a correct matrix Q of eigenvectors, `[Q, V] = refineig(Q, W, B)` replaces any W by the correct eigenvalue matrix V immediately.

- Given a non–defective diagonal sum B of 1-by-1 and 2-by-2 matrices, some with repeated diagonal entries, `[Q, V] = refineig(I, O, B)` produces a correct eigensystem [Q, V] .

- Starting from a slightly perturbed eigensystem [Q, V] of a matrix B with well-separated eigenvalues, the iteration `[Q, V] = refineig(Q, V, B)` converges cubically.

- Starting from a slightly perturbed eigensystem [Q, V] of a non–defective matrix B with well-separated pairs of coincident eigenvalues, the iteration `[Q, V] = refineig(Q, V, B)` converges quadratically.

These expectations were tested upon a number of Hermitian and non–Hermitian matrices, both real and complex, diverse enough to exercize every branch in the program both ways. After these tests had run to satisfactory conclusions, the MATLAB .m-file `refineig.m` was deemed a correct implementation of the intended algorithm. Subsequent tests were designed to shed light upon questions like the following:

- How much does `refineig` improve `eig`'s results in typical non-pathological cases?

- What kinds of pathologies can `refineig` handle that `eig` cannot?

- How much difference does extra-precise accumulation make to `refineig` ?

What is a pathology? It is something to be blamed when results are substantially less accurate than might reasonably have been expected; but appearances can deceive. Roundoff during the generation of a test matrix B can alter its eigensystem enough to swamp the error that `eig` may commit, in which case `eig` might be blamed for getting very nearly the right answer to the wrong question. To prevent this kind of confusion, test data should ideally be free from roundoff before `eig` receives it, although that kind of data consists mainly of small integers quite unlike typical data. On the other hand, `eig` deserves blame if its eigenvalues, but not `refineig`'s, change substantially when its datum B is altered in ways that should not change eigenvalues nor their sensitivities to perturbation; two such alterations are reversal of rows and columns, and transposition. All non-Hermitian test data B were subjected to both reversal and transposition, Hermitian data only to reversal, to see whether substantial consequences ensued.

==========

§A: Acknowledgments

I am grateful to Dr. Cleve Moler for MATLAB, and to Dr. Ali Sazegari of Apple and to Jon Marshall of Intel for computers upon which advantages of extra-precise accumulation of scalar products could be demonstrated. This research was funded in part by U.S. Department of Energy contract DE–FG03–94ER25219, NSF contract ASC-9313958, and NSF Infrastructure Grants Nos. CDA–8722788 and CDA–9401156. Information presented here does not necessarily reflect the position nor the policy of the U.S. Government; no official endorsement should be inferred.

==========

§R: References

E. Bodewig [1959] *Matrix Calculus* 2d. ed., North Holland Publ., Amsterdam; pp. 329-336.

J.W. Demmel [1988] "The Probability that a Numerical Analysis Problem is Difficult" *Math. of Comp.* **50** pp. 449-480. (Set $N = n^2$ and $M = n(n–1)$ in equation (4.12) on p. 458.)

J.W. Demmel & B. Kågström [1993] "The Generalized Schur Decomposition ... Parts I and II" *ACM Trans. Math. Software* **19**, pp. 160-201.

J.J. Dongarra, C.B. Moler & J.H. Wilkinson [1983] "Improving the Accuracy of Computed Eigenvalues and Eigenvectors" *Soc. Indust. Appl. Math. Jl. Numer. Anal.* **20**, pp. 23-45.

P.J. Eberlein [1971] "A note on the matrices denoted by B_n" *Soc. Indust. Appl. Math. Jl. Appl. Math.* **20**, pp. 87-92.

A. Edelman, E. Elmroth & B. Kågström [1997] "A Geometric Approach to Perturbation Theory of Matrices and Matrix Pencils. Parts I and II." *Soc. Indust. Appl. Math. Jl. Matrix Anal. Appl.* **18**, pp. 653-692, and to appear.

W.L. Frank [1958] "Computing eigenvalues of complex matrices by determinant evaluation and by methods of Danilewski and Wielandt" *Jl. Soc. Indust. Appl. Math.* **6**, pp. 378-392, esp. 385, 388.

G.H. Golub & J.H. Wilkinson [1976] "Ill-conditioned eigensystems and the computation of the Jordan canonical form" *Soc. Indust. Appl. Math. Review* **18**, pp. 578-619 (§ 13).

Ming Gu [1995] "Finding Well-Conditioned Similarities to Block-Diagonalize Nonsymmetric Matrices is NP-Hard" *Journal of Complexity* **11**, pp. 377-391.

HP-15C [1982] *Advanced Functions Handbook*, Hewlett-Packard part no. 00015-90011, pp. 148-154.

W. Kahan [1999] "Is there a Skew Cayley Transform with Zero Diagonal?" posted on `http://www.cs.berkeley.edu/~wkahan/skcayley.pdf` .

MATLAB [1993] Version 4.2 of this software was issued by the MathWorks Inc., Natick MA 01760. An earlier version 3.5 was issued in 1991. Versions of `refineig` exist for both versions of MATLAB. The results presented in this opus were obtained from both versions of MATLAB on an Apple Quadra 950 in which a Motorola 68040 does the arithmetic. The later MATLAB version 5.3 released early in 1990 gets the same results on the Quadra but no longer accumulates matrix products extra-precisely on Intel-based PCs and their clones, alas.

J.M. Varah [1986] "A generalization of the Frank matrix" *Soc. Indust. Appl. Math. Jl. Sci. Stat. Comput.* **7**, pp. 835-839.

M. Abramowitz & I.A. Stegun [1964] eds. *Handbook of Math. Functions* ... Nat'l Bureau of St'ds Appl. Math. Series **55**; ch. 22 "Orthog. Polynomials" by U.W. Hochstrasser, §22.7.

J.H. Wilkinson [1960] "Error analysis of floating-point computation" *Numer. Math.* **2**, pp. 319-340 (§ 8).

J.H. Wilkinson [1965] *The Algebraic Eigenvalue Problem*, Oxford U. Press, pp. 92-93.

==========

§F: Appendix on Werner Frank's Matrices

A family of n-by-n matrices discovered serendipitously by Werner L. Frank [1958] have come to be used widely (cf. Wilkinson [1960, 1965] and Golub [1976]) to test eigensystem–computing software because some of their eigenvalues become extremely ill-conditioned as n increases. Properties of these matrices are summarized here. Most properties mentioned have been established mathematically by Eberlein [1971] and by Varah [1986], among others; but a few of these properties, marked by (†) , are inferred only from numerical experiments. The following MATLAB program to compute Frank matrices was adapted from N.J. Higham's:

```
function  F = Frank(n, k)
%     F = Frank(n, k)  is the  Frank  matrix of order  n .  The default
%     is  k = 0 ;  otherwise the elements of  F  get reflected about the
%     anti-diagonal  (1,n)--(n,1) .  Anyway,  Frank  is upper-Hessenberg.

   if nargin == 1, k = 0; end
   if  n < 2,   %  ...  necessitated by  MATLAB's  faulty  diag([], -1) .
         F = eye(n) ;   % ...  with error message if  n < 0 .
    else
         p = [n:-1:1] ;
         F = triu( p( ones(n,1), : ) - diag( ones(n-1,1), -1 ), -1 ) ;
         if k ~= 0 ,  F = F(p,p)';  end, end
```

The following 5-by-5 examples exhibit F = `Frank(5)` , F^T , P = `Frank(5, 1)'` and P^T :

$$F = \begin{bmatrix} 5&4&3&2&1\\ 4&4&3&2&1\\ 0&3&3&2&1\\ 0&0&2&2&1\\ 0&0&0&1&1 \end{bmatrix} , \quad F' = \begin{bmatrix} 5&4&0&0&0\\ 4&4&3&0&0\\ 3&3&3&2&0\\ 2&2&2&2&1\\ 1&1&1&1&1 \end{bmatrix} , \quad P = \begin{bmatrix} 1&1&0&0&0\\ 1&2&2&0&0\\ 1&2&3&3&0\\ 1&2&3&4&4\\ 1&2&3&4&5 \end{bmatrix} \quad \text{and} \quad P' = \begin{bmatrix} 1&1&1&1&1\\ 1&2&2&2&2\\ 0&2&3&3&3\\ 0&0&3&4&4\\ 0&0&0&4&5 \end{bmatrix} .$$

P is obtained by reversing the rows and columns of F . For each dimension n , all four of F , F^T , P and P^T have the same eigenvalues with the same condition numbers, but eigensystem software will compute their smaller eigenvalues with very different accuracies, the more so as n increases. All four matrices have determinant 1 but, for reasons Frank [1958] explained, computed values of $\det(F^T)$ are usually very different from 1 when n is large.

The eigenvalues turn out all positive and they occur in reciprocal pairs; if f is an eigenvalue then $1/f$ is another except that, when n is odd, $f = 1$ is an eigenvalue for which F^T has the row–eigenvector […, 0, –1/3840, 0, 1/384, 0, –1/48, 0, 1/8, 0, –1/2, 0, 1] . (Denominators are products of consecutive even integers.) By reducing the eigenproblem for P to an easier eigenproblem for a tridiagonal matrix, we can show every $(\sqrt{f} - 1/\sqrt{f})$ to be a zero of the *Hermite Polynomial*

$$He_n(x) := (-1)^n \cdot \exp(x^2/2) \cdot d^n(\exp(-x^2/2))/dx^n = x \cdot He_{n-1}(x) - (n-1) \cdot He_{n-2}(x) .$$

(Cf. Abramowitz & Stegun [1964].) Reduction to symmetric tridiagonal form goes as follows:

Let L := `diag(ones(n-1,1), -1)` in MATLAB's notation; this n-by-n lower triangle L has 1 's on the first sub–diagonal and zeros everywhere else. Let W := Diag([0, 1, 2, ..., n–1]) and U := L'·W and V := I + L·U = I+W = Diag([1, 2, 3, ..., n]) . Then we find that

$$P = U + V + L\cdot V + L^2\cdot V + ... + L^{n-1}\cdot V = U + (I–L)^{-1}V = (I–L)^{-1}(I+U) .$$

Now every eigenvalue f of P can be seen to satisfy

$$0 = \det(P - f\cdot I) = \det(I+U – f\cdot(I–L)) = \det(\sqrt{f}\cdot\sqrt{W}\cdot L + (1–f)\cdot I + \sqrt{f}\cdot L'\cdot\sqrt{W})$$

in which the last matrix is obtained from its predecessor via a suitable diagonal similarity. Dividing the last matrix by $\sqrt{f}$ exhibits $\mu := \sqrt{f} - 1/\sqrt{f}$ as an eigenvalue of the symmetric tridiagonal $\sqrt{W}\cdot L + L^T\cdot\sqrt{W}$ whose characteristic polynomial turns out to be

$$\det(\mu\cdot I - \sqrt{W}\cdot L - L^T\cdot\sqrt{W}) = He_n(\mu) .$$

The nonzero zeros of He_n come in pairs $\pm\mu$ to each of which corresponds a pair of eigenvalues $f^{\pm1} = 1 + \mu^2/2 \pm \mu\cdot\sqrt{(1 + \mu^2/4)}$ of P.

Because all diagonal elements vanish in the symmetric tridiagonal matrix, its eigenvalues μ are determined as accurately as are its off-diagonal elements, which are the elements of $\sqrt{W}$. Consequently the eigenvalues f of $P = (I-L)^{-1}(I + L^T\cdot W)$ are determined as accurately as are the elements of W that repeat in the columns of P. However, numerical algorithms that inflict different uncorrelated rounding errors on elements in the same column of P destroy their correlations and, apparently, destroy also the accuracies of the smaller eigenvalues f of P. This reasoning explains why f might not be well-conditioned, but not why it must be so ill–conditioned when computed by conventional software.

The first intimation of ill–condition comes from the disparities between norms of P and its biggest eigenvalues. The Max.-Row-Sum-Norm of P is $n\cdot(n+1)/2$; its Max.–Column–Sum–Norm is $(n\cdot(n+4) - \text{parity}(n))/4$. (Here parity(n) is 0 or 1 according as n is even or odd.) The biggest singular value of P falls short of $(n\cdot(n+4) - 1)/4$ by (†) less than 5% for all n, less than 0.5% for $n > 6$. These are all rather bigger than the biggest eigenvalue f of P, which is smaller than $4\cdot n$ as we shall see next.

For all $n > 1$ the biggest zero μ of He_n lies (†) within 0.5% of

$$g := 2.71983\cdot(\sqrt{n} – \sqrt{3}) – 0.71983\cdot(\sqrt{n + 5.3032} – \sqrt{8.3032}) + \sqrt{3}$$

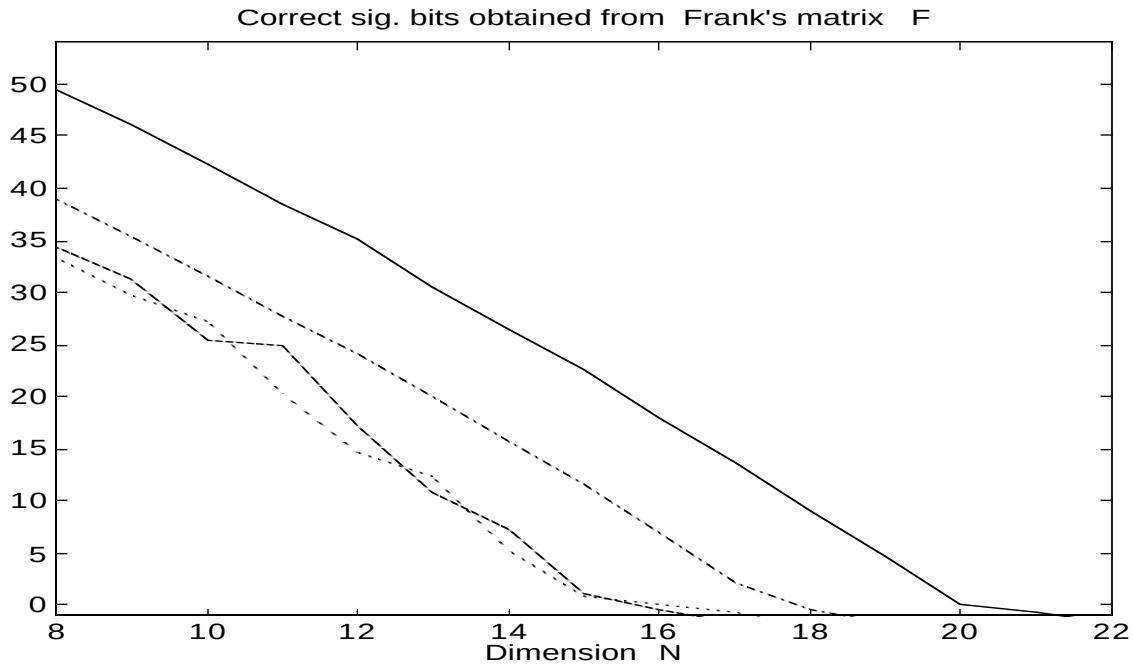
and certainly cannot exceed $\sqrt{n–1} + \sqrt{n–2}$. ($\mu = 0$ at $n = 1$ but $g = 0.008$.) Consequently the extreme eigenvalues $f^{\pm1}$ lie (†) within 1% of $(1 + g^2/2 + g\cdot\sqrt{(1 + g^2/4)})^{\pm1}$ and certainly fall between $(4n)^{\pm1}$.

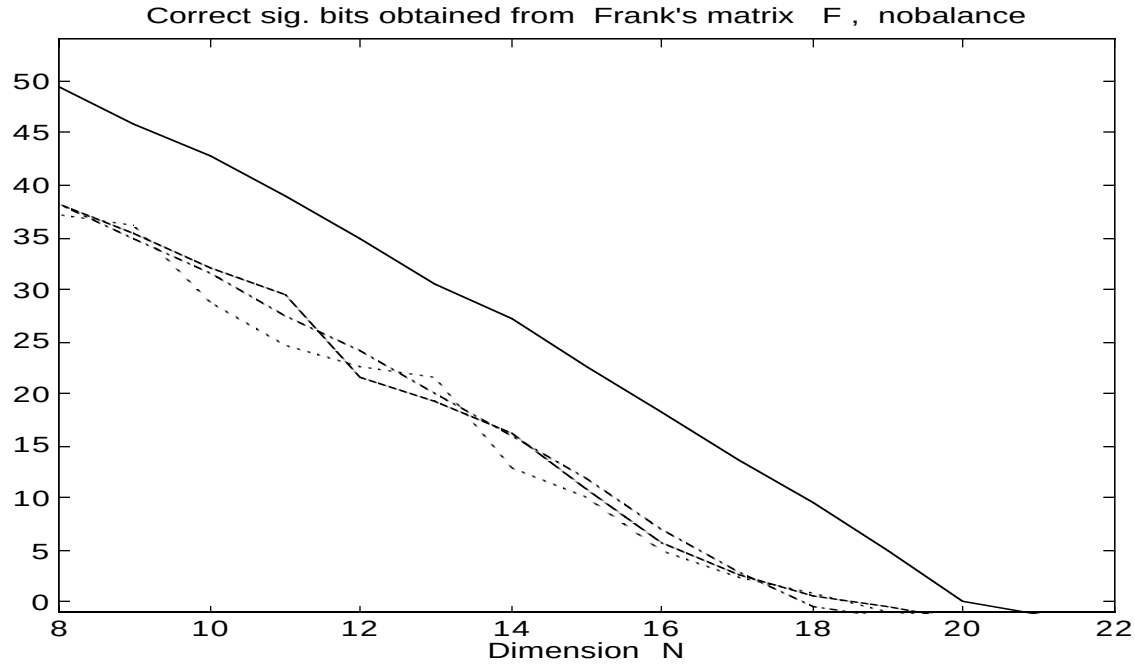
Despite that bigger eigenvalues f become rather smaller than any customary norm of P as n increases, they lose at most a few bits of accuracy to end–figure perturbations of P. But the smallest few eigenvalues f become extremely ill–conditioned. Numerical experiments with $8 \le n \le 19$ indicate (†) that perturbing P to $P + \Delta P$ can lose roughly as many as

$$4.3\,n – 16 \text{ sig. bits}, \quad \text{if} \quad \text{norm}(\Delta P) \le \text{eps}\cdot\text{norm}(P) ,$$
$$4.1\,n – 17 \text{ sig. bits}, \quad \text{if} \quad \text{abs}(\Delta P) \le \text{eps}\cdot\text{abs}(P) \textit{ elementwise},$$

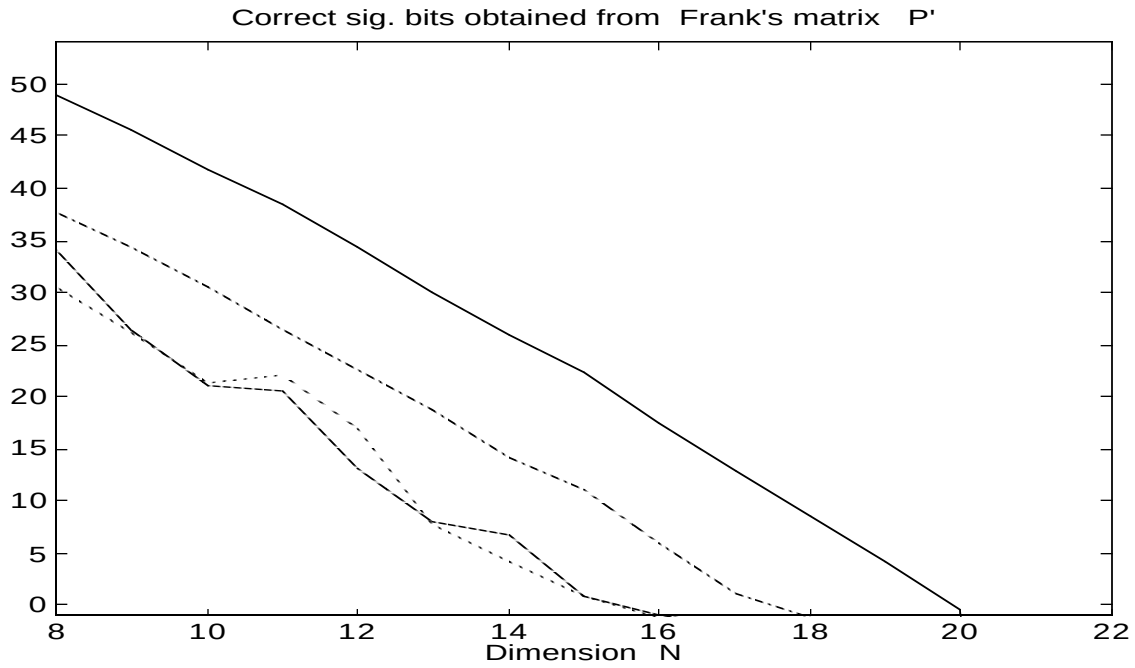
of accuracy in the smallest few eigenvalues; here eps is the roundoff threshold, the difference between 1.0 and the next larger floating-point number 1.000…0001 in the working precision; and norm(…) is the biggest singular value. MATLAB carries 53 sig. bits, so its `eps` $= 2^{–52}$ and, if it has a little bad luck with roundoff, `eig` may deliver utterly inaccurate estimates for those smallest eigenvalues f when $n > 17$.

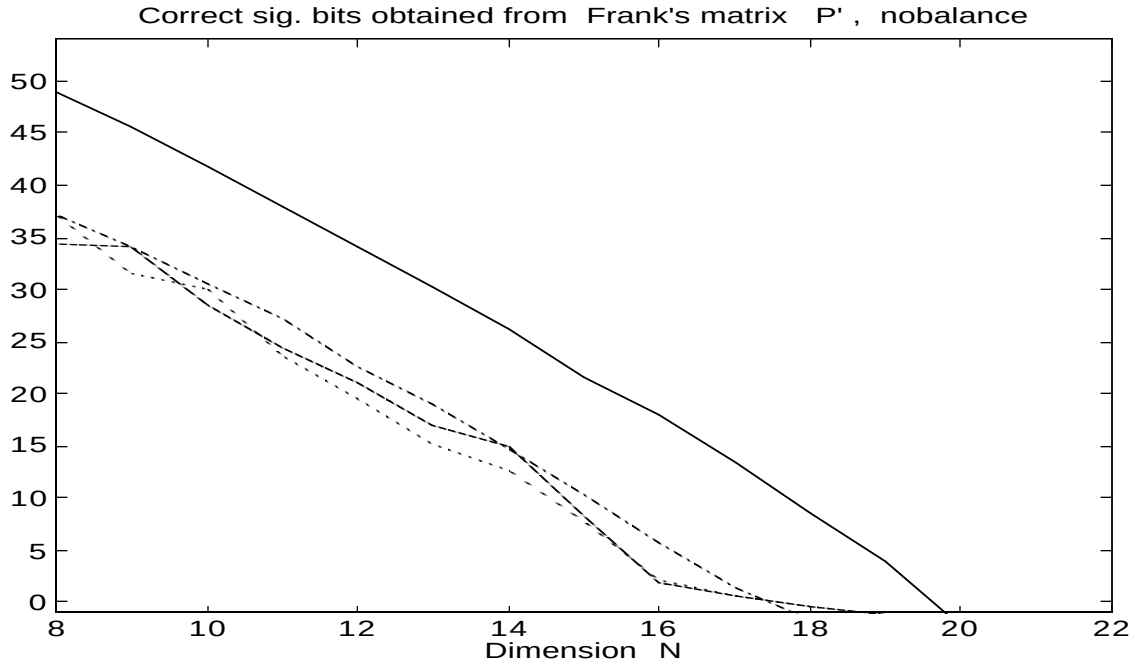
The following results, obtained off my 68040-based Macintosh Quadra 950 , differ negligibly from results off my Pentium-based PC. The results for other computers, such as the MIPS, SPARC, H-P PA, PowerPC/Mac and DEC Alpha, were simulated by setting the Mac's and PC's *Precision Control* to emulate the other computers' arithmetics. The emulation is imperfect, but close enough. These results are from MATLAB 3.5; MATLAB 4.2 does almost the same.
. . . . . .



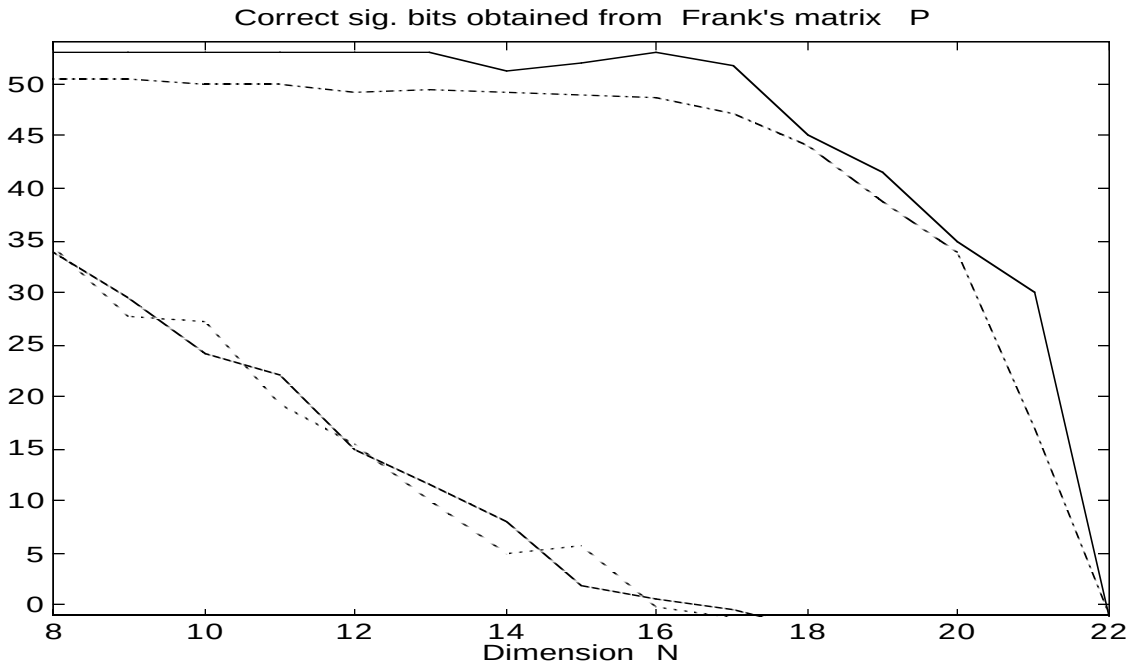


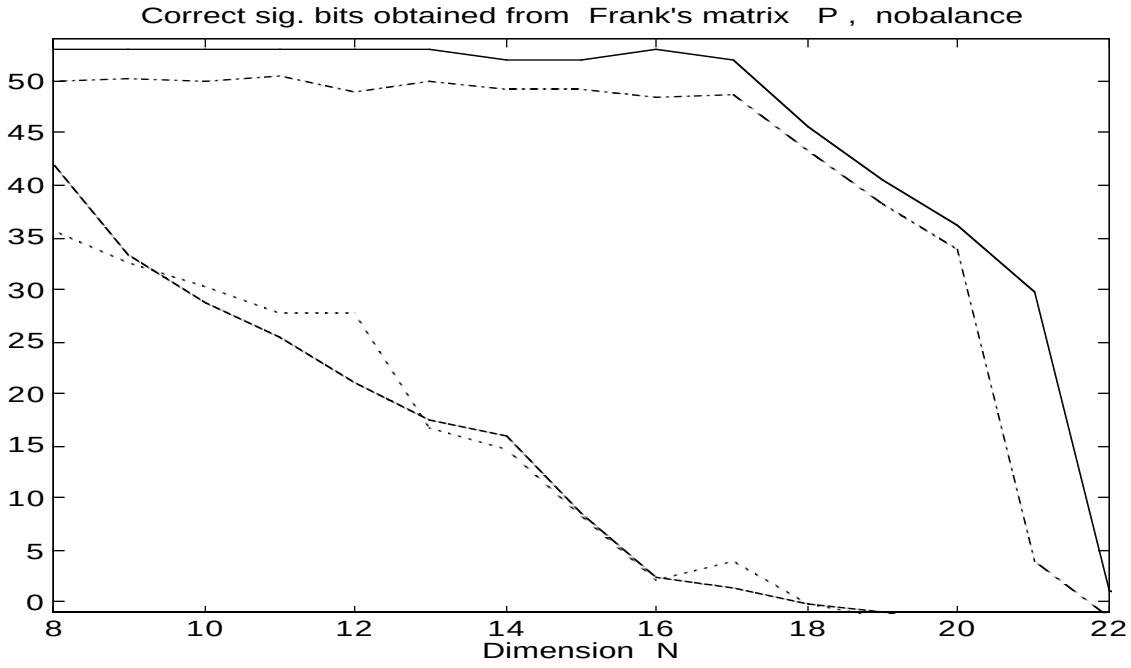
Legend:
- `-----` `eig` on 680x0-Mac or Intel-PC
- `_____` `Refineig` on 680x0-Mac or Intel-PC
- `......` `eig` on others
- `·-·-·-·` `Refineig` on others.



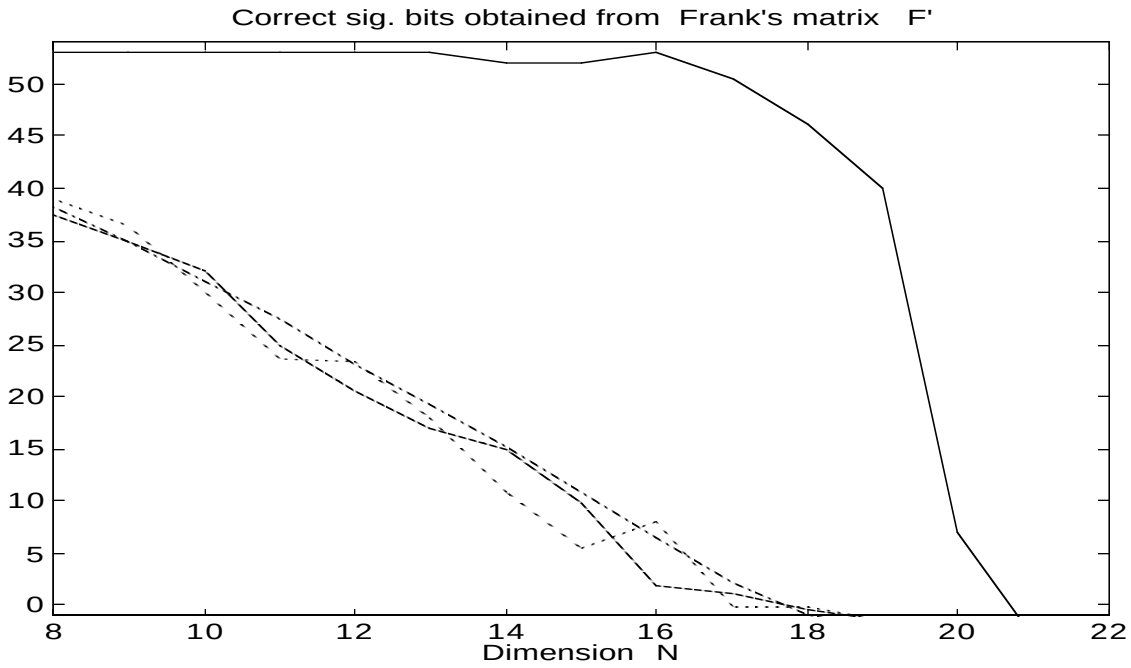


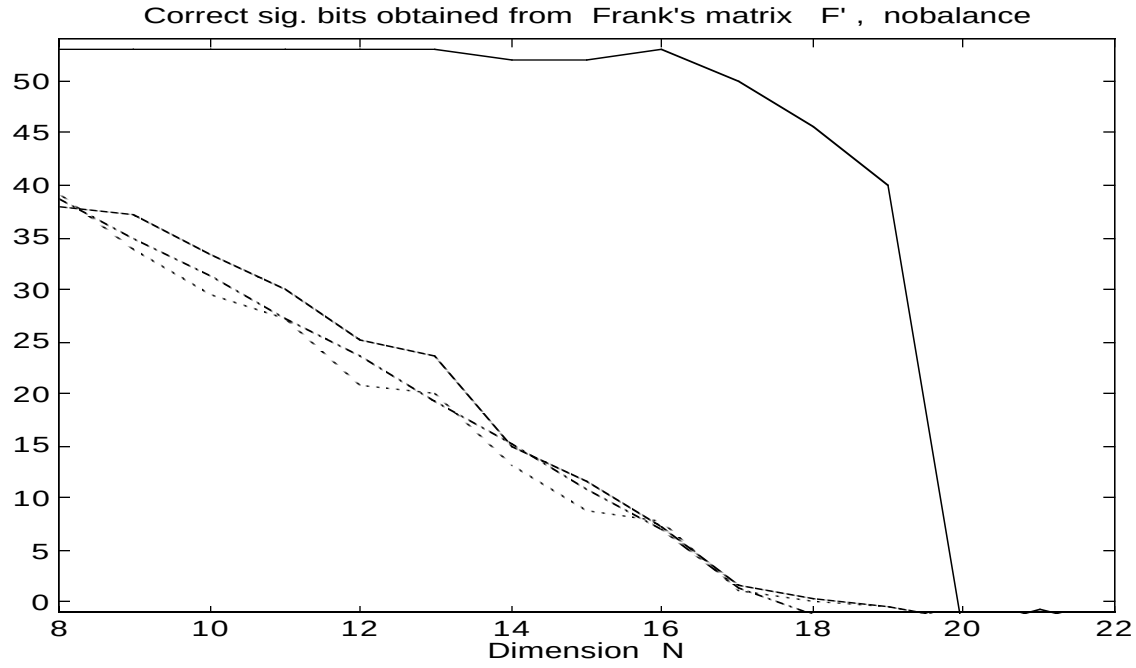
Legend:
- `-----` `eig` on 680x0-Mac or Intel-PC
- `_____` `Refineig` on 680x0-Mac or Intel-PC
- `......` `eig` on others
- `·-·-·-·` `Refineig` on others.





Legend:
- `-----` `eig` on 680x0-Mac or Intel-PC
- `_____` `Refineig` on 680x0-Mac or Intel-PC
- `......` `eig` on others
- `·-·-·-·` `Refineig` on others.





Legend:

-----	`eig`	on 680x0-Mac or Intel-PC
_____	`Refineig`	on 680x0-Mac or Intel-PC
.....	`eig`	on others
·-·-·-·	`Refineig`	on others

§W: Work still to be done:

• How much better than `eig`'s results are `refineig`'s ? How do we know when to quit iterating `refineig` ?
• Explain why balancing Frank matrices is problematical, and to what extent these matrices deserve their notoriety for ill-condition.
• Discuss criteria for adequate tests.
• Can anything be done when the eigenvalues of V+ΔH are disordered, so that part of the correction of the eigenvector-matrix Q involves column swapping? Permutation matrices are all far from I .
• Pursue the examples $B_j(x)$.
• Can anything useful be done for (very nearly) defective matrices?
• Does presubstitution for overflow and division–by–zero really help the computation of ΔZ ?
• MATLAB computes 3/∞ = 0 correctly but bungles $\imath$ /∞ , getting NaN + $\imath$NaN because it converts the latter quotient to $(0 + \imath)*(\infty - \imath 0)/\infty$. The same phenomenon causes $\imath*\infty$ to turn into $(0 + \imath)*(\infty + \imath 0)$ --> NaN + $\imath\infty$. For lack of access to exception-signalling flags, arrays have to be tested element-by-element for ∞ and NaN even though these almost never occur.
• Test cases requiring attention:

Complex Hermitian (Old versions of MATLAB didn't ensure orthogonal eigenvectors)

Norm(Q'·Q – I) smaller than about √eps , dQ := (Q'·Q – I)/2 ;
or smaller than about √√eps , dQ := dQ + (2.5·dQ·dQ – 1.5·dQ)·dQ ;
or not so small , use SVD to get new Q and dQ .
(Test them with B = V = I .)

Computing $f(\Delta S)$ with NaNs and ∞'s in ΔS .

$(I + |dZ|^2)^{-1} \approx I - |dZ|^2$ when $|dZ|^2$ is tiny enough;
use iterative refinement otherwise.

Non-Hermitian

First dZ has NaNs or ∞'s .
Second dZ has NaNs or ∞'s .
Normalization of Q by addition, or by division.
Creation of well-conditioned similarities using $(I - UV')^{-1} = I + U(I - V'U)^{-1}V'$.

Other considerations:
• Modernized Jacobi's iteration may work better on Hermitian matrices than `refineig` can.
• Should `refineig` choose automatically between `refinv` and `refinr` according to whether extra-precise accumulation is available to compute residuals?
• Is there a graceful way to cope with exactly multiple eigenvalues?
• Similar but different algorithms refine Schur factorization, SVD, ...
Clean up notation for Transpose vs. Complex-Conjugate Transpose: B' vs. B' ? B^T vs. B^H ?

§E: An interesting example:

Let $G := \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & -1 & -2 & 1 & -1 & -1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$; its Jordan Normal Form is $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$.

G has four maximal irreducible invariant subspaces for two triple eigenvalues, 0 and 2 , each associated with one 1-dimensional and one 2-dimensional irreducible invariant subspace.

When MATLAB's `eig(G)` is invoked it produces a diagonal matrix of the correct eigenvalues but produces two repetitions of one of at least one of the eigenvalues' eigenvectors; which one depends upon the computer's arithmetic. Since `eig` is not designed to handle matrices with non-diagonal Jordan Normal Forms (even after roundoff incurred by reduction to Hessenberg and then Schur forms), we should not be surprised that `eig` has overlooked one or two eigenvectors. Because the alleged eigenvector matrix is exactly singular — it has an obvious 1- or 2-dimensional nullspace, `refineig` malfunctions too.

But when G is perturbed by subtracting MATLAB's `eps` from G's lower left corner (any random perturbation of magnitude `eps` would probably work almost as well), four eigenvalues are perturbed by about $\sqrt{\texttt{eps}}$, leaving one each of 0 and 2 unperturbed. Then two applications of `refineig` on an Apple Mac Quadra 950 reveal four obviously independent eigenvectors, two for each cluster of eigenvalues. Here they are:

$$Z = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & -2 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 0 & -2 \end{bmatrix} \quad \text{satisfies} \quad G\cdot Z - Z\cdot \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} = O \, .$$