Section 4

Using Matrix Operations

Matrix algebra is a powerful tool. It allows you to more easily formulate and solve many complicated problems, simplifying otherwise intricate computations. In this section you will find information about how the HP-15C performs certain matrix operations and about using matrix operations in your applications.

Several results from numerical linear algebra theory are summarized in this section. This material is not meant to be self-contained. You may want to consult a reference for more complete presentations.*

Understanding the *LU* Decomposition

The HP-15C can solve systems of linear equations, invert matrices, and calculate determinants. In performing these calculations, the HP-15C transforms a square matrix into a computationally convenient form called the *LU* decomposition of the matrix.

The *LU* decomposition procedure factors a square matrix **A** into the matrix product **LU**. **L** is a lower-triangular matrix† with 1's on its diagonal and with subdiagonal elements (those below the diagonal) between −1 and +1, inclusive. **U** is an upper-triangular matrix.† For example:

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ .5 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & -.5 \end{bmatrix} = \mathbf{LU}.$$

* Two such references are
Atkinson, Kendall E., *An Introduction to Numerical Analysis*, Wiley, 1978.
Kahan, W. "Numerical Linear Algebra," *Canadian Mathematical Bulletin*, Volume 9, 1966, pp. 756-801.

† A lower-triangular matrix has 0's for all elements above its diagonal. An upper-triangular matrix has 0's for all elements below its diagonal.

Some matrices can't be factored into the *LU* form. For example,

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \neq \mathbf{LU}$$

for *any* pair of lower- and upper-triangular matrices **L** and **U**. However, if rows are interchanged in the matrix to be factored, an *LU* decomposition can always be constructed. Row interchanges in the matrix **A** can be represented by the matrix product **PA** for some square matrix **P**. Allowing for row interchanges, the *LU* decomposition can be represented by the equation **PA** = **LU**. So for the above example,

$$\mathbf{PA} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} = \mathbf{LU}.$$

Row interchanges can also reduce rounding errors that can occur during the calculation of the decomposition.

The HP-15C uses the Doolittle method with extended-precision arithmetic to construct the *LU* decomposition. It generates the decomposition entirely within the result matrix. The *LU* decomposition is stored in the form

$$\begin{bmatrix} & \mathbf{U} \\ \mathbf{L} & \end{bmatrix}$$

It is not necessary to save the diagonal elements of **L** since they are always equal to 1. The row interchanges are also recorded in the same matrix in a coded form not visible to you. The decomposition is flagged in the process, and its descriptor includes two dashes when displayed.

When you calculate a determinant or solve a system of equations, the *LU* decomposition is automatically saved. It may be useful to use the decomposed form of a matrix as input to a subsequent calculation. If so, it is essential that you not destroy the information about row interchanges stored in the matrix; don't modify the matrix in which the decomposition is stored.

To calculate the determinant of a matrix, **A** for example, the HP-15C uses the equation $\mathbf{A} = \mathbf{P}^{-1}\mathbf{LU}$, which allows for row interchanges. The determinant is then just $(-1)^r$ times the product of the diagonal elements of **U**, where r is the number of row interchanges. The HP-15C calculates this product with the correct sign after decomposing the matrix. If the matrix is already decomposed, the calculator just computes the signed product.

It's easier to invert an upper- or lower-triangular matrix than a general square matrix. The HP-15C calculates the inverse of a matrix, **A** for example, using the relationship

$$\mathbf{A}^{-1} = (\mathbf{P}^{-1}\mathbf{LU})^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}.$$

It does this by first decomposing matrix **A**, inverting both **L** and **U**, calculating their product $\mathbf{U}^{-1}\mathbf{L}^{-1}$, and then interchanging the columns of the result. This is all done within the result matrix—which could be **A** itself. If **A** is already in decomposed form, the decomposition step is skipped. Using this method, the HP-15C can invert a matrix without using additional storage registers.

Solving a system of equations, such as solving $\mathbf{AX} = \mathbf{B}$ for **X**, is easier with an upper- or lower-triangular system matrix **A** than with a general square matrix **A**. Using $\mathbf{PA} = \mathbf{LU}$, the equivalent problem is solving $\mathbf{LUX} = \mathbf{PB}$ for **X**. The rows of **B** are interchanged in the same way that the rows of the matrix **A** were during decomposition. The HP-15C solves $\mathbf{LY} = \mathbf{PB}$ for **Y** (forward substitution) and then $\mathbf{UX} = \mathbf{Y}$ for **X** (backward substitution). The *LU* form is preserved so that you can find the solutions for several matrices **B** without reentering the system matrix.

The *LU* decomposition is an important intermediate step for calculating determinants, inverting matrices, and solving linear systems. The *LU* decomposition can be used in lieu of the original matrix as input to these calculations.

Ill-Conditioned Matrices and the Condition Number

In order to discuss errors in matrix calculations, it's useful to define a measure of distance between two matrices. One measure of the distance between matrices **A** and **B** is the *norm* of their difference, denoted $\|\mathbf{A} - \mathbf{B}\|$. The norm can also be used to define the *condition number* of a matrix, which indicates how the relative error of a calculation compares to the relative error of the matrix itself.

The HP-15C provides three norms. The *Frobenius norm* of a matrix **A**, denoted $\|\mathbf{A}\|_F$, is the square root of the sum of the squares of the matrix elements. This is the matrix analog of the Euclidean length of a vector.

Another norm provided by the HP-15C is the *row norm*. The row norm of an $m \times n$ matrix **A** is the largest row sum of absolute values and is denoted $\|\mathbf{A}\|_R$:

$$\|\mathbf{A}\|_R = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|.$$

The *column norm* of the matrix is denoted $\|\mathbf{A}\|_C$ and can be computed by $\|\mathbf{A}\|_C = \|\mathbf{A}^T\|_R$. The column norm is the largest column sum of absolute values.

For example, consider the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 9 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 2 & 2 & 2 \\ 4 & 5 & 6 \end{bmatrix}.$$

Then

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

and

$$\|\mathbf{A} - \mathbf{B}\|_F = \sqrt{11} \approx 3.3 \text{ (Frobenius norm)},$$
$$\|\mathbf{A} - \mathbf{B}\|_R = 3 \text{ (row norm), and}$$
$$\|\mathbf{A} - \mathbf{B}\|_C = 4 \text{ (column norm).}$$

The remainder of this discussion assumes that the row norm is used. Similar results are obtained if any of the other norms is used instead.

The *condition number* of a square matrix **A** is defined as

$$K(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|.$$

Then $1 \le K(\mathbf{A}) < \infty$ using any norm. The condition number is useful for measuring errors in calculations. A matrix is said to be *ill-conditioned* if $K(\mathbf{A})$ is very large.

If rounding or other errors are present in matrix elements, these errors will propagate through subsequent matrix calculations. They can be magnified significantly. For example, suppose that **X** and **B** are nonzero vectors satisfying **AX** = **B** for some square matrix **A**. Suppose **A** is perturbed by $\Delta\mathbf{A}$ and we compute $\mathbf{B} + \Delta\mathbf{B} = (\mathbf{A} + \Delta\mathbf{A})\mathbf{X}$. Then

$$\frac{(\|\Delta\mathbf{B}\| / \|\mathbf{B}\|)}{(\|\Delta\mathbf{A}\| / \|\mathbf{A}\|)} \leqslant K(\mathbf{A}),$$

with equality for some perturbation $\Delta\mathbf{A}$. This measures how much the relative uncertainty in **A** can be magnified when propagated into the product.

The condition number also measures how much larger in norm the relative uncertainty of the solution to a system can be compared to that of the stored data. Suppose again that **X** and **B** are nonzero vectors satisfying **AX** = **B** for some matrix **A**. Suppose now that matrix **B** is perturbed (by rounding errors, for example) by an amount $\Delta\mathbf{B}$. Let $\mathbf{X} + \Delta\mathbf{X}$ satisfy $\mathbf{A}(\mathbf{X} + \Delta\mathbf{X}) = \mathbf{B} + \Delta\mathbf{B}$. Then

$$\frac{(\|\Delta\mathbf{X}\| / \|\mathbf{X}\|)}{(\|\Delta\mathbf{B}\| / \|\mathbf{B}\|)} \leqslant K(\mathbf{A}),$$

with equality for some perturbation $\Delta\mathbf{B}$.

Suppose instead that matrix **A** is perturbed by $\Delta\mathbf{A}$. Let $\mathbf{X} + \Delta\mathbf{X}$ satisfy $(\mathbf{A} + \Delta\mathbf{A})(\mathbf{X} + \Delta\mathbf{X}) = \mathbf{B}$. If $d(\mathbf{A},\Delta\mathbf{A}) = K(\mathbf{A})\|\Delta\mathbf{A}\| / \|\mathbf{A}\| < 1$, then

$$\frac{(\|\Delta\mathbf{X}\| / \|\mathbf{X}\|)}{(\|\Delta\mathbf{A}\| / \|\mathbf{A}\|)} \leqslant K(\mathbf{A})/(1 - d(\mathbf{A},\Delta\mathbf{A})).$$

Similarly, if $\mathbf{A}^{-1} + \mathbf{Z}$ is the inverse of the perturbed matrix $\mathbf{A} + \Delta\mathbf{A}$, then

$$\frac{(\|\mathbf{Z}\| / \|\mathbf{A}^{-1}\|)}{(\|\Delta\mathbf{A}\| / \|\mathbf{A}\|)} \leqslant K(\mathbf{A})/(1 - d(\mathbf{A},\Delta\mathbf{A})).$$

Moreover, certain perturbations $\Delta\mathbf{A}$ cause the inequalities to become equalities.

All of the preceding relationships show how the relative error of the result is related to the relative error of matrix **A** via the condition number $K(\mathbf{A})$. For each inequality, there are matrices for which equality is true. A large condition number makes possible a relatively large error in the result.

Errors in the data—sometimes very small relative errors—can cause the solution of an ill-conditioned system to be quite different from the solution of the original system. In the same way, the inverse of a perturbed ill-conditioned matrix can be quite different from the inverse of the unperturbed matrix. But both differences are bounded by the condition number; they can be relatively large *only* if the condition number $K(\mathbf{A})$ is large.

Also, a large condition number $K(\mathbf{A})$ of a nonsingluar matrix **A** indicates that the matrix **A** is relatively close, in norm, to a singular matrix. That is.

$$1/K(\mathbf{A}) = \min(\|\mathbf{A} - \mathbf{S}\|/\|\mathbf{A}\|)$$

and

$$1/\|\mathbf{A}^{-1}\| = \min(\|\mathbf{A} - \mathbf{S}\|),$$

where the minimum is taken over all singular matrices **S**. That is, if $K(\mathbf{A})$ is large, then the relative difference between **A** and the closest singular matrix **S** is small. If the norm of $\mathbf{A}^{-1}$ is large, the difference between **A** and the closest singular matrix **S** is small.

For example, let

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & .9999999999 \end{bmatrix}.$$

Then

$$\mathbf{A}^{-1} = \begin{bmatrix} -9{,}999{,}999{,}999 & 10^{10} \\ 10^{10} & -10^{10} \end{bmatrix}$$

and $\|\mathbf{A}^{-1}\| = 2 \times 10^{10}$. Therefore, there should exist a perturbation $\Delta\mathbf{A}$ with $\|\Delta\mathbf{A}\| = 5 \times 10^{-11}$ that makes $\mathbf{A} + \Delta\mathbf{A}$ singular. Indeed, if

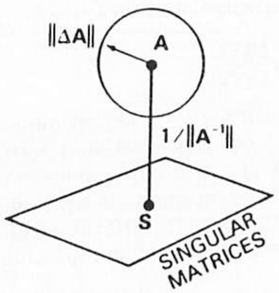
$$\Delta\mathbf{A} = \begin{bmatrix} 0 & -5 \times 10^{-11} \\ 0 & 5 \times 10^{-11} \end{bmatrix}$$

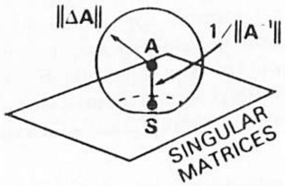
with $\|\Delta\mathbf{A}\| = 5 \times 10^{-11}$, then

$$\mathbf{A} + \Delta\mathbf{A} = \begin{bmatrix} 1 & .99999999995 \\ 1 & .99999999995 \end{bmatrix}$$

and $A + \Delta A$ is singular.

The figures below illustrate these ideas. In each figure matrix A and matrix S are shown relative to the "surface" of singular matrices and within the space of all matrices. Distance is measured using the norm. Around every matrix A is a region of matrices that are practically indistinguishable from A (for example, those within rounding errors of A). The radius of this region is $\|\Delta A\|$. The distance from a nonsingular matrix A to the nearest singular matrix S is $1/\|A^{-1}\|$.





In the left diagram, $\|\Delta A\| < 1/\|A^{-1}\|$. If $\|\Delta A\| << 1/\|A^{-1}\|$ (or $K(A)\|\Delta A\|/\|A\| << 1$), then

$$\begin{aligned}\text{relative variation in } A^{-1} &= \|\text{change in } A^{-1}\|/\|A^{-1}\| \\ &\approx (\|\Delta A\|/\|A\|)K(A) \\ &= \|\Delta A\|/(1/\|A^{-1}\|) \\ &= (\text{radius of sphere})/(\text{distance to surface})\end{aligned}$$

In the right diagram, $\|\Delta A\| > 1/\|A^{-1}\|$. In this case, there exists a singular matrix that is indistinguishable from A, and it may not even be reasonable to try to compute the inverse of A.

The Accuracy of Numerical Solutions to Linear Systems

The preceding discussion dealt with how uncertainties in the data are reflected in the solutions of systems of linear equations and in matrix inverses. But even when data is exact, uncertainties are introduced in numerically calculated solutions and inverses.

Consider solving the linear system $AX = B$ for the theoretical solution X. Because of rounding errors during the calculations, the calculated solution Z is in general not the solution to the original system $AX = B$, but rather the solution to the perturbed system $(A + \Delta A)Z = B$. The perturbation ΔA satisfies $\|\Delta A\| \leq \epsilon\|A\|$, where ϵ is usually a very small number. In many cases, ΔA will amount to less than one in the 10th digit of each element of A.

For a calculated solution Z, the *residual* is $R = B - AZ$. Then $\|R\| \leq \epsilon\|A\|\|Z\|$. So the expected residual for a calculated solution is small. But although the residual R is usually small, the *error* $Z - X$ may not be small if A is ill-conditioned:

$$\|Z - X\| \leq \epsilon\|A\|\|A^{-1}\|\|Z\| = \epsilon K(A)\|Z\|.$$

A useful rule-of-thumb for the accuracy of the computed solution is

$$\begin{pmatrix}\text{number of correct}\\ \text{decimal digits}\end{pmatrix} \geq \begin{pmatrix}\text{number of}\\ \text{digits carried}\end{pmatrix} - \log(\|A\|\|A^{-1}\|) - \log(10n)$$

where n is the dimension of A. For the HP-15C, which carries 10 accurate digits,

$$(\text{number of correct decimal digits}) \geq 9 - \log(\|A\|\|A^{-1}\|) - \log(n).$$

In many applications, this accuracy may be adequate. When additional accuracy is desired, the computed solution Z can usually be improved by *iterative refinement* (also known as *residual correction*).

Iterative refinement involves calculating a solution to a system of equations, then improving its accuracy using the residual associated with the solution to modify that solution.

To use iterative refinement, first calculate a solution Z to the original system $AX = B$. Z is then treated as an approximation to

X, in error by $\mathbf{E} = \mathbf{X} - \mathbf{Z}$. Then **E** satisfies the linear system $\mathbf{AE} = \mathbf{AX} - \mathbf{AZ} = \mathbf{R}$, where **R** is the residual for **Z**. The next step is to calculate the residual and then to solve $\mathbf{AE} = \mathbf{R}$ for **E**. The calculated solution, denoted by **F**, is treated as an approximation to $\mathbf{E} = \mathbf{X} - \mathbf{Z}$ and is added to **Z** to obtain a new approximation to **X**: $\mathbf{F} + \mathbf{Z} \approx (\mathbf{X} - \mathbf{Z}) + \mathbf{Z} = \mathbf{X}$.

In order for $\mathbf{F} + \mathbf{Z}$ to be a better approximation to **X** than is **Z**, the residual $\mathbf{R} = \mathbf{B} - \mathbf{AZ}$ must be calculated to extended precision. The HP-15C's MATRIX 6 operation does this. The system matrix **A** is used for finding both solutions, **Z** and **F**. The *LU* decomposition formed while calculating **Z** can be used for calculating **F**, thereby shortening the execution time. The refinement process can be repeated, but most of the improvement occurs in the first refinement.

(Refer to Applications at the end of this section for a program that performs one iteration of refinement.)

Making Difficult Equations Easier

A system of equations $\mathbf{EX} = \mathbf{B}$ is difficult to numerically solve accurately if **E** is ill-conditioned (nearly singular). Even iterative refinement can fail to improve the calculated solution when **E** is sufficiently ill-conditioned. However, instances arise in practice when a modest extra effort suffices to change difficult equations into others with the same solution, but which are easier to solve. Scaling and preconditioning are two processes to do this.

Scaling

Bad scaling is a common cause of poor results from attempts to numerically invert ill-conditioned matrices or to solve systems of equations with ill-conditioned system matrices. But it is a cause that you can easily diagnose and cure.

Suppose a matrix **E** is obtained from a matrix **A** by $\mathbf{E} = \mathbf{LAR}$, where **L** and **R** are scaling diagonal matrices whose diagonal elements are all integer powers of 10. Then **E** is said to be obtained from **A** by *scaling*. **L** scales the rows of **A**, and **R** scales the columns. Presumably $\mathbf{E}^{-1} = \mathbf{R}^{-1}\mathbf{A}^{-1}\mathbf{L}^{-1}$ can be obtained either from $\mathbf{A}^{-1}$ by scaling or from **E** by inverting.

For example, let matrix **A** be

$$\mathbf{A} = \begin{bmatrix} 3 \times 10^{-40} & 1 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & -1 \end{bmatrix}.$$

The HP-15C correctly calculates $\mathbf{A}^{-1}$ to 10-digit accuracy as

$$\mathbf{A}^{-1} \approx \begin{bmatrix} -2 & 3 & -1 \\ 3 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}.$$

Now let

$$\mathbf{L} = \mathbf{R} = \begin{bmatrix} 10^{20} & 0 & 0 \\ 0 & 10^{-20} & 0 \\ 0 & 0 & 10^{-20} \end{bmatrix}$$

so that

$$\mathbf{E} = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 10^{-40} & 10^{-40} \\ 2 & 10^{-40} & -10^{-40} \end{bmatrix}.$$

E is very near a singular matrix

$$\mathbf{S} = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

and $\|\mathbf{E} - \mathbf{S}\| / \|\mathbf{E}\| = \frac{1}{3} \times 10^{-40}$. This means that $K(\mathbf{S}) \geqslant 3 \times 10^{40}$, so it's not surprising that the calculated $\mathbf{E}^{-1}$

$$\mathbf{E}^{-1} \approx \begin{bmatrix} -6.67 \times 10^{-11} & 1 & 10^{-10} \\ 0.8569 & 8.569 \times 10^{9} & -4.284 \times 10^{9} \\ 0.07155 & -4.284 \times 10^{9} & 2.142 \times 10^{9} \end{bmatrix}$$

is far from the true value

$$\mathbf{E}^{-1} = \begin{bmatrix} -2\times 10^{-40} & 3 & -1 \\ 3 & -4\times 10^{40} & 2\times 10^{40} \\ -1 & 2\times 10^{40} & -10^{40} \end{bmatrix}.$$

Multiplying the calculated inverse and the original matrix verifies that the calculated inverse is poor.

The trouble is that **E** is badly scaled. A well-scaled matrix, like **A**, has all its rows and columns comparable in norm *and* the same must hold true for its inverse. The rows and columns of **E** are about as comparable in norm as those of **A**, but the first row and column of $\mathbf{E}^{-1}$ are small in norm compared with the others. Therefore, to achieve better numerical results, the rows and columns of **E** should be scaled before the matrix is inverted. This means that the diagonal matrices **L** and **R** discussed earlier should be chosen to make **LER** and $(\mathbf{LER})^{-1} = \mathbf{R}^{-1}\mathbf{E}^{-1}\mathbf{L}^{-1}$ not so badly scaled.

In general, you can't know the true inverse of matrix **E** in advance. So the detection of bad scaling in **E** and the choice of scaling matrices **L** and **R** must be based on **E** and the *calculated* $\mathbf{E}^{-1}$. The calculated $\mathbf{E}^{-1}$ shows poor scaling and might suggest trying

$$\mathbf{L} = \mathbf{R} = \begin{bmatrix} 10^{-5} & 0 & 0 \\ 0 & 10^{5} & 0 \\ 0 & 0 & 10^{5} \end{bmatrix}.$$

Using these scaling matrices,

$$\mathbf{LER} = \begin{bmatrix} 3\times 10^{-10} & 1 & 2 \\ 1 & 10^{-30} & 10^{-30} \\ 2 & 10^{-30} & -10^{-30} \end{bmatrix},$$

which is still poorly scaled, but not so poorly that the HP-15C can't cope. The calculated inverse is

$$(\mathbf{LER})^{-1} = \begin{bmatrix} -2\times 10^{-30} & 3 & -1 \\ 3 & -4\times 10^{30} & 2\times 10^{30} \\ -1 & 2\times 10^{30} & -10^{30} \end{bmatrix}.$$

This result is correct to 10 digits, although you wouldn't be expected to know this. This result is verifiably correct in the sense that using the calculated inverse,

$$(\mathbf{LER})^{-1}(\mathbf{LER}) = (\mathbf{LER})(\mathbf{LER})^{-1} = \mathbf{I} \text{ (the identity matrix)}$$

to 10 digits.

Then $\mathbf{E}^{-1}$ is calculated as

$$\mathbf{E}^{-1} = \mathbf{R}(\mathbf{LER})^{-1}\mathbf{L} = \begin{bmatrix} -2\times 10^{-40} & 3 & -1 \\ 3 & -4\times 10^{40} & 2\times 10^{40} \\ -1 & 2\times 10^{40} & -10^{40} \end{bmatrix},$$

which is correct to 10 digits.

If $(\mathbf{LER})^{-1}$ is verifiably poor, you can repeat the scaling, using **LER** in place of **E** and using new scaling matrices suggested by **LER** and the calculated $(\mathbf{LER})^{-1}$.

You can also apply scaling to solving a system of equations, for example **EX** = **B**, where **E** is poorly scaled. When solving for **X**, replace the system **EX** = **B** by a system (**LER**)**Y** = **LB** to be solved for **Y**. The diagonal scaling matrices **L** and **R** are chosen as before to make the matrix **LER** well-scaled. After you calculate **Y** from the new system, calculate the desired solution as **X** = **RY**.

Preconditioning

Preconditioning is another method by which you can replace a difficult system, **EX** = **B**, by an easier one, **AX** = **D**, with the same solution **X**.

Suppose that **E** is ill-conditioned (nearly singular). You can detect this by calculating the inverse $\mathbf{E}^{-1}$ and observing that $1/\|\mathbf{E}^{-1}\|$ is very small compared to $\|\mathbf{E}\|$ (or equivalently by a large condition number $K(\mathbf{E})$). Then almost every row vector $\mathbf{u}^T$ will have the property that $\|\mathbf{u}^T\| / \|\mathbf{u}^T\mathbf{E}^{-1}\|$ is also very small compared with $\|\mathbf{E}\|$, where $\mathbf{E}^{-1}$ is the calculated inverse. This is because most row vectors $\mathbf{u}^T$ will have $\|\mathbf{u}^T\mathbf{E}^{-1}\|$ not much smaller than $\|\mathbf{u}^T\|\|\mathbf{E}^{-1}\|$, and $\|\mathbf{E}^{-1}\|$ will be large. Choose such a row vector $\mathbf{u}^T$ and calculate $\mathbf{v}^T = a\mathbf{u}^T\mathbf{E}^{-1}$. Choose the scalar a so that the row vector $\mathbf{r}^T$, obtained by rounding every element of $\mathbf{v}^T$ to an integer between -100 and 100, does not differ much from $\mathbf{v}^T$. Then $\mathbf{r}^T$ is a row vector with integer elements with magnitudes less than 100. $\|\mathbf{r}^T\mathbf{E}\|$ will be small compared with $\|\mathbf{r}^T\|\|\mathbf{E}\|$—the smaller the better.

Next, choose the kth element of $\mathbf{r}^T$ having one of the largest magnitudes. Replace the kth row of **E** by $\mathbf{r}^T\mathbf{E}$ and the kth row of **B** by $\mathbf{r}^T\mathbf{B}$. Provided that no roundoff has occurred during the evaluation of these new rows, the new system matrix **A** should be better conditioned (farther from singular) than **E** was, but the system will still have the same solution **X** as before.

This process works best when **E** and **A** are both scaled so that every row of **E** and of **A** have roughly the same norm as every other. You can do this by multiplying the rows of the systems of equations **EX** = **B** and **AX** = **D** by suitable powers of 10. If **A** is not far enough from singular, though well scaled, repeat the preconditioning process.

As an illustration of the preconditioning process, consider the system **EX** = **B**, where

$$\mathbf{E} = \begin{bmatrix} x & y & y & y & y \\ y & x & y & y & y \\ y & y & x & y & y \\ y & y & y & x & y \\ y & y & y & y & x \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and $x = 8000.00002$ and $y = -1999.99998$. If you attempt to solve this system directly, the HP-15C calculates the solution **X** and the inverse $\mathbf{E}^{-1}$ to be

$$\mathbf{X} \approx \begin{bmatrix} 2014.6 \\ 2014.6 \\ 2014.6 \\ 2014.6 \\ 2014.6 \end{bmatrix} \text{ and } \mathbf{E}^{-1} \approx 2014.6 \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Substituting, you find

$$\mathbf{EX} \approx \begin{bmatrix} 1.00146 \\ 0.00146 \\ 0.00146 \\ 0.00146 \\ 0.00147 \end{bmatrix}.$$

Upon checking (using $\boxed{\text{MATRIX}}$ 7), you find that $1/\|\mathbf{E}^{-1}\| \approx 9.9 \times 10^{-5}$, which is very small compared with $\|\mathbf{E}\| \approx 1.6 \times 10^4$ (or that the calculated condition number is large—$\|\mathbf{E}\|\,\|\mathbf{E}^{-1}\| \approx 1.6 \times 10^8$).

Choose any row vector $\mathbf{u}^T = (1, 1, 1, 1, 1)$ and calculate

$$\mathbf{u}^T\mathbf{E}^{-1} \approx 10{,}073\,(1, 1, 1, 1, 1).$$

Using $a = 10^{-4}$,

$$\begin{aligned} \mathbf{v}^T &= a\,\mathbf{u}^T\mathbf{E}^{-1} \approx 1.0073\,(1, 1, 1, 1, 1) \\ \mathbf{r}^T &= (1, 1, 1, 1, 1) \\ \|\mathbf{r}^T\mathbf{E}\| &\approx 5 \times 10^{-4} \\ \|\mathbf{r}^T\|\,\|\mathbf{E}\| &\approx 8 \times 10^4. \end{aligned}$$

As expected, $\|\mathbf{r}^T\mathbf{E}\|$ is small compared with $\|\mathbf{r}^T\|\|\mathbf{E}\|$.

Now replace the first row of **E** by

$$10^7\mathbf{r}^T\mathbf{E} = (1000, 1000, 1000, 1000, 1000)$$

and the first row of **B** by $10^7\mathbf{r}^T\mathbf{B} = 10^7$. This gives a new system equation **AX** = **D**, where

$$\mathbf{A} = \begin{bmatrix} 1000 & 1000 & 1000 & 1000 & 1000 \\ y & x & y & y & y \\ y & y & x & y & y \\ y & y & y & x & y \\ y & y & y & y & x \end{bmatrix} \text{ and } \mathbf{D} = \begin{bmatrix} 10^7 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Note that $\mathbf{r}^T\mathbf{E}$ was scaled by 10^7 so that each row of $\mathbf{E}$ and $\mathbf{A}$ has roughly the same norm as every other. Using this new system, the HP-15C calculates the solution

$$\mathbf{X} = \begin{bmatrix} 2000.000080 \\ 1999.999980 \\ 1999.999980 \\ 1999.999980 \\ 1999.999980 \end{bmatrix}, \text{ with } \mathbf{AX} = \begin{bmatrix} 10^7 \\ -10^{-5} \\ -9 \times 10^{-6} \\ 0 \\ 0 \end{bmatrix}.$$

This solution differs from the earlier solution and is correct to 10 digits.

Sometimes the elements of a nearly singular matrix $\mathbf{E}$ are calculated using a formula to which roundoff contributes so much error that the calculated inverse $\mathbf{E}^{-1}$ must be wrong even when it is calculated using exact arithmetic. Preconditioning is valuable in this case only if it is applied to the formula in such a way that the modified row of $\mathbf{A}$ is calculated accurately. In other words, you must change the formula exactly into a new and better formula by the preconditioning process if you are to gain any benefit.

Least-Squares Calculations

Matrix operations are frequently used in *least-squares* calculations. The typical least-squares problem involves an $n \times p$ matrix $\mathbf{X}$ of observed data and a vector $\mathbf{y}$ of n observations from which you must find a vector $\mathbf{b}$ with p coefficients that minimizes

$$\|\mathbf{r}\|_F^2 = \sum_{i=1}^{n} r_i^2$$

where $\mathbf{r} = \mathbf{y} - \mathbf{Xb}$ is the residual vector.

Normal Equations

From the expression above,

$$\|\mathbf{r}\|_F^2 = (\mathbf{y} - \mathbf{Xb})^T(\mathbf{y} - \mathbf{Xb}) = \mathbf{y}^T\mathbf{y} - 2\mathbf{b}^T\mathbf{X}^T\mathbf{y} + \mathbf{b}^T\mathbf{X}^T\mathbf{Xb}.$$

Solving the least-squares problem is equivalent to finding a solution $\mathbf{b}$ to the *normal equations*

$$\mathbf{X}^T\mathbf{Xb} = \mathbf{X}^T\mathbf{y}.$$

However, the normal equations are very sensitive to rounding errors. (Orthogonal factorization, discussed on page 113, is relatively insensitive to rounding errors.)

The *weighted least-squares* problem is a generalization of the ordinary least-squares problem. In it you seek to minimize

$$\|\mathbf{Wr}\|_F^2 = \sum_{i=1}^{n} w_i^2 r_i^2$$

where $\mathbf{W}$ is a diagonal $n \times n$ matrix with positive diagonal elements $w_1, w_2, \ldots, w_n$.

Then

$$\|\mathbf{Wr}\|_F^2 = (\mathbf{y} - \mathbf{Xb})^T\mathbf{W}^T\mathbf{W}(\mathbf{y} - \mathbf{Xb})$$

and any solution $\mathbf{b}$ also satisfies the weighted normal equations

$$\mathbf{X}^T\mathbf{W}^T\mathbf{WXb} = \mathbf{X}^T\mathbf{W}^T\mathbf{Wy}.$$

These are the normal equations with $\mathbf{X}$ and $\mathbf{y}$ replaced by $\mathbf{WX}$ and $\mathbf{Wy}$. Consequentially, these equations are sensitive to rounding errors also.

The *linearly constrained least-squares* problem involves finding $\mathbf{b}$ such that it minimizes

$$\|\mathbf{r}\|_F^2 = \|\mathbf{y} - \mathbf{Xb}\|_F^2$$

subject to the constraints

$$\mathbf{Cb} = \mathbf{d} \qquad \left(\sum_{j=1}^{k} c_{ij} b_j = d_i \text{ for } i = 1, 2, \ldots, m\right).$$

This is equivalent to finding a solution $\mathbf{b}$ to the *augmented normal equations*

$$\begin{bmatrix} \mathbf{X}^T\mathbf{X} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{l} \end{bmatrix} = \begin{bmatrix} \mathbf{X}^T\mathbf{y} \\ \mathbf{d} \end{bmatrix}$$

where $\mathbf{l}$, a vector of Lagrange multipliers, is part of the solution but isn't used further. Again, the augmented equations are very sensitive to rounding errors. Note also that weights can also be included by replacing $\mathbf{X}$ and $\mathbf{y}$ with $\mathbf{WX}$ and $\mathbf{Wy}$.

As an example of how the normal equations can be numerically unsatisfactory for solving least-squares problems, consider the system defined by

$$\mathbf{X} = \begin{bmatrix} 100{,}000. & -100{,}000. \\ 0.1 & 0.1 \\ 0.2 & 0.0 \\ 0.0 & 0.2 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}.$$

Then

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 10{,}000{,}000{,}000.05 & -9{,}999{,}999{,}999.99 \\ -9{,}999{,}999{,}999.99 & 10{,}000{,}000{,}000.05 \end{bmatrix}$$

and

$$\mathbf{X}^T\mathbf{y} = \begin{bmatrix} 10{,}000.03 \\ -9{,}999.97 \end{bmatrix}.$$

However, when rounded to 10 digits,

$$\mathbf{X}^T\mathbf{X} \approx \begin{bmatrix} 10^{10} & -10^{10} \\ -10^{10} & 10^{10} \end{bmatrix},$$

which is the same as what would be calculated if $\mathbf{X}$ were rounded to five significant digits relative to the largest element:

$$\mathbf{X} = \begin{bmatrix} 100{,}000 & -100{,}000 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

The HP-15C solves $\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$ (perturbing the singular matrix as described on page 118) and gets

$$\mathbf{b} = \begin{bmatrix} 0.060001 \\ 0.060000 \end{bmatrix}$$

with

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{b} = \begin{bmatrix} 0.03 \\ 0.03 \end{bmatrix}.$$

However, the correct least-squares solution is

$$\mathbf{b} = \begin{bmatrix} 0.5000005 \\ 0.4999995 \end{bmatrix}$$

despite the fact that the calculated solution and the exact solution satisfy the computed normal equations equally well.

The normal equations should be used only when the elements of $\mathbf{X}$ are all small integers (say between -3000 and 3000) or when you know that no perturbations in the columns $\mathbf{x}_j$ of $\mathbf{X}$ of as much as $\|\mathbf{x}_j\|/10^4$ could make those columns linearly dependent.

Orthogonal Factorization

The following orthogonal factorization method solves the least-squares problem and is less sensitive to rounding errors than the normal equation method. You might use this method when the normal equations aren't appropriate.

Any $n \times p$ matrix $\mathbf{X}$ can be factored as $\mathbf{X} = \mathbf{Q}^T\mathbf{U}$, where $\mathbf{Q}$ is an $n \times n$ orthogonal matrix characterized by $\mathbf{Q}^T = \mathbf{Q}^{-1}$ and $\mathbf{U}$ is an $n \times p$ upper-triangular matrix. The essential property of orthogonal matrices is that they preserve length in the sense that

$$\begin{aligned} \|\mathbf{Qr}\|_F^2 &= (\mathbf{Qr})^T(\mathbf{Qr}) \\ &= \mathbf{r}^T\mathbf{Q}^T\mathbf{Qr} \\ &= \mathbf{r}^T\mathbf{r} \\ &= \|\mathbf{r}\|_F^2. \end{aligned}$$

Therefore, if $\mathbf{r} = \mathbf{y} - \mathbf{Xb}$, it has the same length as

$$\mathbf{Qr} = \mathbf{Qy} - \mathbf{QXb} = \mathbf{Qy} - \mathbf{Ub}.$$

The upper-triangular matrix $\mathbf{U}$ and the product $\mathbf{Qy}$ can be written as

$$\mathbf{U} = \begin{bmatrix} \hat{\mathbf{U}} \\ \mathbf{0} \end{bmatrix} \begin{array}{l} (p \text{ rows}) \\ (n-p \text{ rows}) \end{array} \quad \text{and} \quad \mathbf{Qy} = \begin{bmatrix} \mathbf{g} \\ \mathbf{f} \end{bmatrix} \begin{array}{l} (p \text{ rows}) \\ (n-p \text{ rows}) \end{array}$$

Then

$$\begin{aligned} \|\mathbf{r}\|_F^2 &= \|\mathbf{Qr}\|_F^2 \\ &= \|\mathbf{Qy} - \mathbf{Ub}\|_F^2 \\ &= \|\mathbf{g} - \hat{\mathbf{U}}\mathbf{b}\|_F^2 + \|\mathbf{f}\|_F^2 \\ &\geqslant \|\mathbf{f}\|_F^2 \end{aligned}$$

with equality when $\mathbf{g} - \hat{\mathbf{U}}\mathbf{b} = \mathbf{0}$. In other words, the solution to the ordinary least-squares problem is any solution to $\hat{\mathbf{U}}\mathbf{b} = \mathbf{g}$ and the minimal sum of squares is $\|\mathbf{f}\|_F^2$. This is the basis of all numerically sound least-squares programs.

You can solve the unconstrained least-squares problem in two steps:

1. Perform the orthogonal factorization of the augmented $n \times (p+1)$ matrix

$$\begin{bmatrix} \mathbf{X} & \mathbf{y} \end{bmatrix} = \mathbf{Q}^T\mathbf{V}$$

where $\mathbf{Q}^T = \mathbf{Q}^{-1}$, and retain only the upper-triangular factor $\mathbf{V}$, which you can then partition as

$$\mathbf{V} = \begin{array}{cc} \begin{bmatrix} \hat{\mathbf{U}} & \mathbf{g} \\ \mathbf{0} & q \\ \mathbf{0} & \mathbf{0} \end{bmatrix} & \begin{array}{l} (p \text{ rows}) \\ (1 \text{ row}) \\ (n-p-1 \text{ rows}) \end{array} \\ \begin{array}{cc} \uparrow & \uparrow \\ (p \text{ columns}) & (1 \text{ column}) \end{array} & \end{array}$$

Only the first $p + 1$ rows (and columns) of $\mathbf{V}$ need to be retained. (Note that $\mathbf{Q}$ here is not the same as that mentioned earlier, since this $\mathbf{Q}$ must also transform $\mathbf{y}$.)

2. Solve the following system for $\mathbf{b}$:

$$\begin{bmatrix} \hat{\mathbf{U}} & \mathbf{g} \\ 0 & q \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -q \end{bmatrix}.$$

(If $q = 0$, replace it by any small nonzero number, say 10^{-99}.) The -1 in the solution matrix automatically appears; it requires no additional calculations.

In the absence of rounding errors, $q = \pm\|\mathbf{y} - \mathbf{Xb}\|_F$; this may be inaccurate if $|q|$ is too small, say smaller than $\|\mathbf{y}\|/10^6$. If you desire a more accurate estimate of $\|\mathbf{y} - \mathbf{Xb}\|_F$, you can calculate it directly from $\mathbf{X}$, $\mathbf{y}$, and the computed solution $\mathbf{b}$.

For the weighted least-squares problem, replace $\mathbf{X}$ and $\mathbf{y}$ by $\mathbf{WX}$ and $\mathbf{Wy}$, where $\mathbf{W}$ is the diagonal matrix containing the weights.

For the linearly constrained least-squares problem, you must recognize that constraints may be inconsistent. In addition, they can't always be satisfied exactly by a calculated solution because of rounding errors. Therefore, you must specify a tolerance t such that the constraints are said to be satisfied when $\|\mathbf{Cb} - \mathbf{d}\| < t$. Certainly $t > \|\mathbf{d}\|/10^{10}$ for 10-digit computation, and in some cases a much larger tolerance must be used.

Having chosen t, select a weight factor w that satisfies $w > \|\mathbf{y}\|/t$. For convenience, choose w to be a power of 10 somewhat bigger than $\|\mathbf{y}\|/t$. Then $w\|\mathbf{Cb} - \mathbf{d}\| > \|\mathbf{y}\|$ unless $\|\mathbf{Cb} - \mathbf{d}\| < t$.

However, the constraint may fail to be satisfied for one of two reasons:

- No $\mathbf{b}$ exists for which $\|\mathbf{Cb} - \mathbf{d}\| < t$.
- The leading columns of $\mathbf{C}$ are nearly linearly dependent.

Check for the first situation by determining whether a solution exists for the constraints alone. When $[w\mathbf{C} \quad w\mathbf{d}]$ has been factored to $\mathbf{Q}[\mathbf{U} \quad \mathbf{g}]$, solve this system for $\mathbf{b}$

$$\begin{array}{r} (k \text{ rows}) \\ (p+1-k \text{ rows}) \end{array} \begin{bmatrix} \mathbf{U} & \mathbf{g} \\ \mathbf{0} & \operatorname{diag}(q) \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -q \end{bmatrix} \begin{array}{l} (p \text{ rows}) \\ (1 \text{ row}) \end{array}$$

using any small nonzero number q. If the computed solution $\mathbf{b}$ satisfies $\mathbf{Cb} \approx \mathbf{d}$, then the constraints are not inconsistent.

The second situation is rarely encountered and can be avoided. It shows itself by causing at least one of the diagonal elements of **U** to be much smaller than the largest element above it in the same column, where **U** is from the orthogonal factorization $w\mathbf{C} = \mathbf{QU}$.

To avoid this situation, reorder the columns of $w\mathbf{C}$ and **X** and similarly reorder the elements (rows) of **b**. The reordering can be chosen easily if the troublesome diagonal element of **U** is also much smaller than some subsequent element in its row. Just swap the corresponding columns in the original data and refactor the weighted constraint equations. Repeat this procedure if necessary.

For example, if the factorization of $w\mathbf{C}$ gives

$$\mathbf{U} = \begin{bmatrix} 1.0 & 2.0 & 0.5 & -1.5 & 0.3 \\ 0 & 0.02 & 0.5 & 3.0 & 0.1 \\ 0 & 0 & 2.5 & 1.5 & -1.2 \end{bmatrix},$$

then the second diagonal element is much smaller than the value 2.0 above it. This indicates that the first and second columns in the original constraints are nearly dependent. The diagonal element is also much smaller than the subsequent value 3.0 in its row. Then the second and fourth columns should be swapped in the original data and the factorization repeated.

It is always prudent to check for consistent constraints. The test for small diagonal elements of **U** can be done at the same time.

Finally, using **U** and **g** as the first k rows, add rows corresponding to **X** and **y**. (Refer to Least-Squares Using Successive Rows on page 140 for additional information.) Then solve the unconstrained least-squares problem with

$$\mathbf{X} \rightarrow \begin{bmatrix} w\mathbf{C} \\ \mathbf{X} \end{bmatrix} \quad \text{and} \quad \mathbf{y} \rightarrow \begin{bmatrix} w\mathbf{d} \\ \mathbf{y} \end{bmatrix}.$$

Provided the calculated solution **b** satisfies $\|\mathbf{Cb} - \mathbf{d}\| < t$, that solution will also minimize $\|\mathbf{y} - \mathbf{Xb}\|$ subject to the constraint $\mathbf{Cb} \approx \mathbf{d}$.

Singular and Nearly Singular Matrices

A matrix is singular if and only if its determinant is zero. The determinant of a matrix is equal to $(-1)^r$ times the product of the diagonal elements of **U**, where **U** is the upper-diagonal matrix of the matrix's LU decomposition and r is the number of row interchanges in the decomposition. Then, theoretically, a matrix is singular if at least one of the diagonal elements of **U**, the pivots, is zero; otherwise it is nonsingular.

However, because the HP-15C performs calculations with only a finite number of digits, some singular and nearly singular matrices can't be distinguished in this way. For example, consider the matrix

$$\mathbf{B} = \begin{bmatrix} 3 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 0 & 0 \end{bmatrix} = \mathbf{LU},$$

which is singular. Using 10-digit accuracy, this matrix is decomposed as

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 \\ .3333333333 & 1 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 0 & 10^{-10} \end{bmatrix},$$

which is nonsingular. The singular matrix **B** can't be distinguished from the nonsingular matrix

$$\mathbf{D} = \begin{bmatrix} 3 & 3 \\ .9999999999 & 1 \end{bmatrix}$$

since they both have identical calculated LU decompositions.

On the other hand, the matrix

$$\mathbf{A} = \begin{bmatrix} 3 & 3 \\ 1 & .9999999999 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 0 & -10^{-10} \end{bmatrix} = \mathbf{LU}$$

is nonsingular. Using 10-digit accuracy, matrix **A** is decomposed as

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 \\ .3333333333 & 1 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 0 & 0 \end{bmatrix}.$$

This would incorrectly indicate that matrix **A** is singular. The nonsingular matrix **A** can't be distinguished from the singular matrix

$$\mathbf{C} = \begin{bmatrix} 3 & 3 \\ .9999999999 & .9999999999 \end{bmatrix}$$

since they both have identical calculated *LU* decompositions.

When you use the HP-15C to calculate an inverse or to solve a system of equations, you should understand that some singular and nearly singular matrices have the same calculated *LU* decomposition. For this reason, the HP-15C *always* calculates a result by ensuring that all decomposed matrices *never* have zero pivots. It does this by perturbing the pivots, if necessary, by an amount that is usually smaller than the rounding error in the calculations. This enables you to invert matrices and solve systems of equations without being interrupted by zero pivots. This is very important in applications such as calculating eigenvectors using the method of inverse iteration (refer to page 155).

The effect of rounding errors and possible intentional perturbations is to cause the calculated decomposition to have all nonzero pivots and to correspond to a nonsingular matrix $\mathbf{A} + \Delta\mathbf{A}$ usually identical to or negligibly different from the original matrix **A**. Specifically, unless every element in some column of **A** has absolute value less than 10^{-89}, the column sum norm $\|\Delta\mathbf{A}\|_C$ will be negligible (to 10 significant digits) compared with $\|\mathbf{A}\|_C$.

The HP-15C calculates the determinant of a square matrix as the signed product of the (possibly perturbed) calculated pivots. The calculated determinant is the determinant of the matrix $\mathbf{A} + \Delta\mathbf{A}$ represented by the *LU* decomposition. It can be zero only if the product's magnitude becomes smaller than 10^{-99} (underflow).

Applications

The following programs illustrate how you can use matrix operations to solve many types of advanced problems.

Constructing an Identity Matrix

This program creates an identity matrix $\mathbf{I}_n$ in the matrix whose descriptor is in the Index register. The program assumes that the matrix is already dimensioned to $n \times n$. Execute the program using GSB 8. The final matrix will have 1's for all diagonal elements and 0's for all other elements.

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	
f LBL 8	001-42,21, 8	
f MATRIX 1	002-42,16, 1	Sets $i = j = 1$.
f LBL 9	003-42,21, 9	
RCL 0	004- 45 0	
RCL 1	005- 45 1	
g TEST 6	006-43,30, 6	Tests $i \neq j$.
g CLx	007- 43 35	
g TEST 5	008-43,30, 5	Tests $i = j$.
EEX	009- 26	Sets element to 1 if $i = j$.
f USER STO (i) f USER	010u 44 24	Skips next step at last element.
GTO 9	011- 22 9	
g RTN	012- 43 32	
g P/R		Run mode.

Labels used: 8 and 9.

Registers used: R_0, R_1, and Index register.

One-Step Residual Correction

The following program solves the system of equations $\mathbf{AX} = \mathbf{B}$ for **X**, then performs one stage iterative refinement to improve the solution. The program uses four matrices:

Appendix

Accuracy of Numerical Calculations

Misconceptions About Errors

Error is not sin, nor is it always a mistake. Numerical error is merely the difference between what you wish to calculate and what you get. The difference matters only if it is too big. Usually it is negligible; but sometimes error is distressingly big, hard to explain, and harder to correct. This appendix focuses on errors, especially those that might be large—however rare. Here are some examples.

Example 1: A Broken Calculator. Since $(\sqrt{x})^2 = x$ whenever $x \geqslant 0$, we expect also

$$f(x) = ((\ldots((\underbrace{\sqrt{\sqrt{\ldots\sqrt{\sqrt{x}}}}}_{\substack{50\\ \text{roots}}})^2)^2 \underbrace{\ldots)^2)^2}_{\substack{50\\ \text{squares}}}$$

should equal x too.

A program of 100 steps can evaluate the expression $f(x)$ for any positive x. When $x = 10$ the HP-15C calculates 1 instead. The error $10 - 1 = 9$ appears enormous considering that only 100 arithmetic operations were performed, each one presumably correct to 10 digits. What the program actually delivers instead of $f(x) = x$ turns out to be

$$f(x) = \begin{cases} 1 & \text{for } x \geqslant 1 \\ 0 & \text{for } 0 \leqslant x < 1, \end{cases}$$

which seems very wrong. Should this calculator be repaired?

Example 2: Many Pennies. A corporation retains Susan as a scientific and engineering consultant at a fee of one penny per second for her thoughts, paid every second of every day for a year. Rather than distract her with the sounds of pennies dropping, the corporation proposes to deposit them for her into a bank account in which interest accrues at the rate of 11¼ percent per annum compounded every second. At year's end these pennies will accumulate to a sum

$$\text{total} = (\text{payment}) \times \frac{(1 + i/n)^n - 1}{i/n}$$

where payment = \$0.01 = one penny per second,

$i = 0.1125 = 11.25$ percent per annum interest rate,

$n = 60 \times 60 \times 24 \times 365$ = number of seconds in a year.

Using her HP-15C, Susan reckons that the total will be \$376,877.67 . But at year's end the bank account is found to hold \$333,783.35 . Is Susan entitled to the \$43,094.32 difference?

In both examples the discrepancies are caused by rounding errors that could have been avoided. This appendix explains how.

The war against error begins with a salvo against wishful thinking, which might confuse what we want with what we get. To avoid confusion, the true and calculated results must be given different names even though their difference may be so small that the distinction seems pedantic.

Example 3: Pi. The constant $\pi = 3.1415926535897932384626433\ldots$ Pressing the $\boxed{\pi}$ key on the HP-15C delivers a different value

$$\boxed{\pi} = 3.141592654$$

which agrees with π to 10 significant digits. But $\boxed{\pi} \neq \pi$, so we should not be surprised when, in Radians mode, the calculator doesn't produce $\sin \boxed{\pi} = 0$.

Suppose we wish to calculate x but we get X instead. (This convention is used throughout this appendix.) The *error* is $x - X$. The *absolute error* is $|x - X|$. The *relative error* is usually reckoned $(x - X)/x$ for $x \neq 0$.

Example 4: A Bridge Too Short. The lengths in meters of three sections of a cantilever bridge are designed to be

$$x = 333.76 \qquad y = 195.07 \qquad z = 333.76\,.$$

The measured lengths turn out to be respectively

$$X = 333.69 \qquad Y = 195.00 \qquad Z = 333.72\,.$$

The discrepancy in total length is

$$d = (x + y + z) - (X + Y + Z) = 862.59 - 862.41 = 0.18\,.$$

Ed, the engineer, compares the discrepancy d with the total length $(x + y + z)$ and considers the relative discrepancy

$$d/(x + y + z) = 0.0002 = 2 \text{ parts in } 10{,}000$$

to be tolerably small. But Rhonda, the riveter, considers the absolute discrepancy $|d| = 0.18$ meters (about 7 inches) much too large for her liking; some powerful stretching will be needed to line up the bridge girders before she can rivet them together. Both see the same discrepancy d, but what looks neglibible to one person can seem awfully big to another.

Whether large or small, errors must have sources which, if understood, usually permit us to compensate for the errors or to circumvent them altogether. To understand the distortions in the girders of a bridge, we should learn about structural engineering and the theory of elasticity. To understand the errors introduced by the very act of computation, we should learn how our calculating instruments work and what are their limitations. These are details most of us want not to know, especially since a well-designed calculator's rounding errors are always nearly minimal and therefore appear insignificant when they are introduced. But when on rare occasions they conspire to send a computation awry, they must be reclassified as "significant" after all.

Example 1 Explained. Here $f(x) = s(r(x))$, where

$$r(x) = \underbrace{\sqrt{\sqrt{\ldots\sqrt{\sqrt{x}}}}}_{\substack{50\\ \text{roots}}} = x^{(\frac{1}{2}^{50})}$$

and

$$s(r) = ((\ldots\underbrace{((r)^2)^2\ldots)^2}_{\substack{50\\ \text{squares}}})^2 = r^{(2^{50})}.$$

The exponents are $\frac{1}{2}^{50} = 8.8818 \times 10^{-16}$ and $2^{50} = 1.1259 \times 10^{15}$. Now, x must lie between 10^{-99} and $9.999\ldots \times 10^{99}$ since no positive numbers outside that range can be keyed into the calculator. Since r is an increasing function, $r(x)$ lies between

$$r(10^{-99}) = 0.9999999999997975\ldots$$

and

$$r(10^{100}) = 1.0000000000002045\ldots\,.$$

This suggests that $R(x)$, the calculated value of $r(x)$, would be 1 for all valid calculator arguments x. In fact, because of roundoff,

$$R(x) = \begin{cases} 0.9999999999 & \text{for } 0 < x < 1 \\ 1.000000000 & \text{for } 1 \leqslant x \leqslant 9.999999999 \times 10^{99}. \end{cases}$$

If $0 < x < 1$, then $x \leqslant 0.9999999999$ in a 10-digit calculator. We would then rightly expect that $\sqrt{x} \leqslant \sqrt{0.9999999999}$, which is $0.99999999994999999999 8\ldots$, which rounds to 0.9999999999 again. Therefore, if $\boxed{\sqrt{x}}$ is pressed arbitrarily often starting with $x < 1$, the result cannot exceed 0.9999999999. This explains why we obtain $R(x) = 0.9999999999$ for $0 < x < 1$ above. When $R(x)$ is squared 50 times to produce $F(x) = S(R(x))$, the result is clearly 1 for $x \geqslant 1$, but why is $F(x) = 0$ for $0 \leqslant x < 1$? When $x < 1$,

$$s(R(x)) \leqslant s(0.9999999999) = (1 - 10^{-10})^{2^{50}} \approx 6.14 \times 10^{-48898}.$$

This value is so small that the calculated value $F(x) = S(R(x))$ underflows to 0. So the HP-15C isn't broken; it is doing the best that can be done with 10 significant digits of precision and 2 exponent digits.

We have explained example 1 using no more information about the HP-15C than that it performs each arithmetic operation $\boxed{\sqrt{x}}$ and $\boxed{x^2}$ fully as accurately as is possible within the limitations of 10 significant digits and 2 exponent digits. The rest of the information we needed was mathematical knowledge about the functions f, r, and s. For instance, the value $r(10^{100})$ above was evaluated as

$$\begin{aligned} r(10^{100}) &= (10^{100})^{(1/2^{50})} \\ &= \exp(\ln(10^{100})/2^{50}) \\ &= \exp(100(\ln 10)/2^{50}) \\ &= \exp(2.045 \times 10^{-13}) \\ &= 1 + (2.045 \times 10^{-13}) + \tfrac{1}{2}(2.045 \times 10^{-13})^2 + \dots \end{aligned}$$

by using the series $\exp(z) = 1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{6}z^3 + \dots$.

Similarly, the binomial theorem was used for

$$\begin{aligned} \sqrt{0.9999999999} &= (1 - 10^{-10})^{1/2} \\ &= 1 - \tfrac{1}{2}(10^{-10}) - \tfrac{1}{8}(10^{-10})^2 - \dots . \end{aligned}$$

These mathematical facts lie well beyond the kind of knowledge that might have been considered adequate to cope with a calculation containing only a handful of multiplications and square roots. In this respect, example 1 illustrates an unhappy truism: Errors make computation very much harder to analyze. That is why a well-designed calculator, like the HP-15C, will introduce errors of its own as sparingly as is possible at a tolerable cost. Much more error than that would turn an already difficult task into something hopeless.

Example 1 should lay two common *misconceptions* to rest:

- Rounding errors can overwhelm a computation only if vast numbers of them accumulate.
- A few rounding errors can overwhelm a computation only if accompanied by massive cancellation.

Regarding the first misconception, example 1 would behave in the same perverse way if it suffered only one rounding error, the one that produces $R(x) = 1$ or 0.9999999999, in error by less than one unit in its last (10th) significant digit.

Regarding the second misconception, cancellation is what happens when two nearly equal numbers are subtracted. For example, calculating

$$c(x) = (1 - \cos x)/x^2$$

in Radians mode for small values of x is hazardous because of cancellation. Using $x = 1.2 \times 10^{-5}$ and rounding results to 10 digits,

$$\cos x = 0.9999999999$$

and

$$1 - \cos x = 0.0000000001$$

with cancellation leaving maybe one significant digit in the numerator. Also

$$x^2 = 1.44 \times 10^{-10}.$$

Then

$$C(x) = 0.6944\,.$$

This calculated value is wrong because $0 \leqslant c(x) < \tfrac{1}{2}$ for all $x \neq 0$. To avoid numerical cancellation, exploit the trigonometric identity $\cos x = 1 - 2\sin^2(x/2)$ to cancel the 1 *exactly* and obtain a better formula

$$c(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

When this latter expression is evaluated (in Radians mode) at $x = 1.2 \times 10^{-5}$, the computed result $C(x) = 0.5$ is correct to 10 significant digits. This example, while explaining the meaning of the word "cancellation," suggests that it is always a bad thing. That is another misconception to be dispatched later. For the present, recall that example 1 contains no subtraction, therefore no cancellation, and is still devastated by its rounding error. In this respect example 1 is counterintuitive, a little bit scary. Nowhere in it can we find one or two arithmetic operations to blame for the catastrophe; no small rearrangement will set everything right as happened for $c(x)$. Alas, example 1 is not an isolated example. As computers and calculators grow in power, so do instances of insidious error growth become more common.

To help you recognize error growth and cope with it is the ultimate goal of this appendix. We shall start with the simplest kinds of errors and work our way up gradually to the subtle errors that can afflict the sophisticated computations possible on the HP-15C.

A Hierarchy of Errors

Some errors are easier to explain and to tolerate than others. Therefore, the functions delivered by single keystrokes on the HP-15C have been categorized, for the purposes of easier exposition, according to how difficult their errors are to estimate. The estimates should be regarded as goals set by the calculator's designers rather than as specifications that guarantee some stated level of accuracy. On the other hand, the designers believe they can prove mathematically that their accuracy goals have been achieved, and extensive testing has produced no indication so far that they might be mistaken.

Level 0: No Error

Functions which should map small integers (smaller than 10^{10}) to small integers do so exactly, without error, as you might expect.

Examples:

$$\sqrt{4}=2 \qquad -2^3=-8 \qquad 3^{20}=3{,}486{,}784{,}401$$

$$\log(10^9)=9 \qquad 6!=720$$

$$\cos^{-1}(0)=90 \text{ (in Degrees mode)}$$

$$\text{ABS}(4{,}684{,}660+4{,}684{,}659\,i)=6{,}625{,}109 \text{ (in Complex mode)}$$

Also exact for real argments are $\boxed{\text{ABS}}$, $\boxed{\text{FRAC}}$, $\boxed{\text{INT}}$, $\boxed{\text{RND}}$, and comparisons (such as $\boxed{x\leq y}$). But the matrix functions $\boxed{\times}$, $\boxed{\div}$, $\boxed{1/x}$, $\boxed{\text{MATRIX}}$6, and $\boxed{\text{MATRIX}}$9 (determinant) are exceptions (refer to page 192).

Level ∞: Overflow/Underflow

Results which would lie closer to zero than 10^{-99} underflow quietly to zero. Any result that would lie beyond the overflow thresholds $\pm 9.999999999\times 10^{99}$ is replaced by the nearest threshold, and then flag 9 is set and the display blinks. (Pressing $\boxed{\text{ON}}\boxed{\text{ON}}$ or $\boxed{\text{CF}}$9 or $\boxed{\leftarrow}$ will clear flag 9 and stop the blinking.) Most functions that result in more than one component can tolerate overflow/underflow in one component without contaminating the other; examples are $\boxed{\to\text{R}}$, $\boxed{\to\text{P}}$, complex arithmetic, and most matrix operations. The exceptions are matrix inversion ($\boxed{1/x}$ and $\boxed{\div}$), $\boxed{\text{MATRIX}}$9 (determinant), and $\boxed{\text{L.R.}}$.

Level 1: Correctly Rounded, or Nearly So

Operations that deliver "correctly rounded" results whose error cannot exceed ½ unit in their last (10th) significant digit include the real algebraic operations $\boxed{+}$, $\boxed{-}$, $\boxed{\times}$, $\boxed{\div}$, $\boxed{x^2}$, $\boxed{\sqrt{x}}$, $\boxed{1/x}$, and $\boxed{\%}$, the complex and matrix operations $\boxed{+}$ and $\boxed{-}$, matrix by scalar operations $\boxed{\times}$ and $\boxed{\div}$ (excluding division by a matrix), and $\boxed{\to\text{H.MS}}$. These results are the best that 10 significant digits can represent, as are familiar constants $\boxed{\pi}$, 1 $\boxed{e^x}$, 2 $\boxed{\text{LN}}$, 10 $\boxed{\text{LN}}$, 1 $\boxed{\to\text{RAD}}$, and many more. Operations that can suffer a slightly larger error, but still significantly smaller than one unit in the 10th significant digit of the result, include $\boxed{\Delta\%}$, $\boxed{\to\text{H}}$, $\boxed{\to\text{RAD}}$, $\boxed{\to\text{DEG}}$, $\boxed{P_{y,x}}$, and $\boxed{C_{y,x}}$; $\boxed{\text{LN}}$, $\boxed{\text{LOG}}$, $\boxed{10^x}$, and $\boxed{\text{TANH}}$ for real arguments; $\boxed{\to\text{P}}$, $\boxed{\text{SIN}^{-1}}$, $\boxed{\text{COS}^{-1}}$, $\boxed{\text{TAN}^{-1}}$, $\boxed{\text{SINH}^{-1}}$, $\boxed{\text{COSH}^{-1}}$, and $\boxed{\text{TANH}^{-1}}$ for real and complex arguments; $\boxed{\text{ABS}}$, $\boxed{\sqrt{x}}$, and $\boxed{1/x}$ for complex arguments; matrix norms $\boxed{\text{MATRIX}}$ 7 and $\boxed{\text{MATRIX}}$ 8; and finally $\boxed{\text{SIN}}$, $\boxed{\text{COS}}$, and $\boxed{\text{TAN}}$ for real arguments in Degrees and Grads modes (but not in Radians mode—refer to Level 2, page 184).

A function that grows to ∞ or decays to 0 exponentially fast as its argument approaches ±∞ may suffer an error larger than one unit in its 10th significant digit, but only if its magnitude is smaller than 10^{-20} or larger than 10^{20}; and though the relative error gets worse as the result gets more extreme (small or large), the error stays below three units in the last (10th) significant digit. The reason for this error is explained later. Functions so affected are $\boxed{e^x}$, $\boxed{y^x}$, $\boxed{x!}$ (for noninteger x), $\boxed{\text{SINH}}$, and $\boxed{\text{COSH}}$ for real arguments. The worst case known is 3^{201}, which is calculated as $7.968419664\times 10^{95}$. The last digit 4 should be 6 instead, as is the case for $7.29^{33.5}$, calculated as $7.968419666\times 10^{28}$.

The foregoing statements about errors can be summarized for all functions in Level 1 in a way that will prove convenient later:

> Attempts to calculate a function f in Level 1 produce instead a computed value $F = (1+\epsilon)f$ whose relative error ϵ, though unknown, is very small:
>
> $$|\epsilon| < \begin{cases} 5 \times 10^{-10} & \text{if } F \text{ is correctly rounded} \\ 1 \times 10^{-9} & \text{for all other functions } F \text{ in Level 1.} \end{cases}$$

This simple characterization of all the functions in Level 1 fails to convey many other important properties they all possess, properties like

- Exact integer values: mentioned in Level 0.
- Sign symmetry: $\sinh(-x) = -\sinh(x)$, $\cosh(-x) = \cosh(x)$, $\ln(1/x) = -\ln(x)$ (if $1/x$ is computed exactly).
- Monotonicity: if $f(x) \geqslant f(y)$, then computed $F(x) \geqslant F(y)$.

These additional properties have powerful implications; for instance, TAN(20°) = TAN(200°) = TAN(2,000°) = ... = TAN(2×10^{99} °) = 0.3639702343 correctly. But the simple characterization conveys most of what is worth knowing, and that can be worth money.

Example 2 Explained. Susan tried to calculate

$$\text{total} = \text{payment} \times \frac{(1+i/n)^n - 1}{i/n}$$

where

payment = \$0.01,

$i = 0.1125$, and

$n = 60 \times 60 \times 24 \times 365 = 31{,}536{,}000$.

She calculated \$376,877.67 on her HP-15C, but the bank's total was \$333,783.35, and this latter total agrees with the results calculated on good, modern financial calculators like the HP-12C, HP-37E, HP-38E/38C, and HP-92. Where did Susan's calculation go awry? No severe cancellation, no vast accumulation of errors; just one rounding error that grew insidiously caused the damage:

$$i/n = 0.0000000035673515 98$$

$$1 + i/n = 1.000000004$$

when rounded to 10 significant digits. There is the rounding error that hurts. Subsequently attempting to calculate $(1+i/n)^n$, Susan must get instead $(1.000000004)^{31{,}536{,}000} = 1.134445516$, which is wrong in its second decimal place.

How can the correct value be calculated? Only by not throwing away so many digits of i/n. Observe that

$$(1+i/n)^n = e^{n\ln(1+i/n)},$$

so we might try to calculate the logarithm in some way that does not discard those precious digits. An easy way to do so on the HP-15C does exist.

To calculate $\lambda(x) = \ln(1+x)$ accurately for all $x > -1$, even if $|x|$ is very small:

1. Calculate $u = 1 + x$ rounded.
2. Then

$$\lambda(x) = \begin{cases} x & \text{if } u = 1 \\ \ln(u)\,x/(u-1) & \text{if } u \neq 1\,. \end{cases}$$

The following program calculates $\lambda(x) = \ln(1+x)$.

Keystrokes	Display	
g P/R		
f CLEAR PRGM	000-	
f LBL A	001-42,21,11	Assumes x is in X-register.
ENTER	002- 36	
ENTER	003- 36	
EEX	004- 26	Places 1 in X-register.
+	005- 40	Calculates $u = 1 + x$ rounded.
g LN	006- 43 12	Calculates $\ln(u)$ (zero for $u = 1$).
x⥂y	007- 34	Restores x to X-register.
g LSTx	008- 43 36	Recalls u.

Keystrokes	Display		
$\boxed{\text{EEX}}$	009-	26	Places 1 in X-register.
$\boxed{\text{g}}\boxed{\text{TEST}}$ 6	010-43,30,	6	Tests $u \neq 1$.
$\boxed{-}$	011-	30	Calculates $u - 1$ when $u \neq 1$.
$\boxed{\div}$	012-	10	Calculates $x/(u-1)$ or $1/1$.
$\boxed{\times}$	013-	20	Calculates $\lambda(x)$.
$\boxed{\text{g}}\boxed{\text{RTN}}$	014-	43 32	
$\boxed{\text{g}}\boxed{\text{P/R}}$			

The calculated value of u, correctly rounded by the HP-15C, is $u = (1 + \epsilon)(1 + x)$, where $|\epsilon| < 5 \times 10^{-10}$. If $u = 1$, then

$$|x| = |1/(1+\epsilon) - 1| \leq 5 \times 10^{-10}$$

too, in which case the Taylor series $\lambda(x) = x\,(1 - \frac{1}{2}x + \frac{1}{3}x^2 - \ldots)$ tells us that the correctly rounded value of $\lambda(x)$ must be just x. Otherwise, we shall calculate $x\,\lambda(u-1)/(u-1)$ fairly accurately instead of $\lambda(x)$. But $\lambda(x)/x = 1 - \frac{1}{2}x + \frac{1}{3}x^2 - \ldots$ varies very slowly, so slowly that the absolute error $\lambda(x)/x - \lambda(u-1)/(u-1)$ is no worse than the absolute error $x - (u - 1) = -\epsilon(1 + x)$, and if $x \leq 1$, this error is negligible relative to $\lambda(x)/x$. When $x > 1$, then $u - 1$ is so nearly x that the error is negligible again; $\lambda(x)$ is correct to nine significant digits.

As usual in error analyses, the explanation is far longer than the simple procedure being explained and obscures an important fact: the errors in $\ln(u)$ and $u - 1$ were ignored during the explanation because we knew they would be negligible. This knowledge, and hence the simple procedure, is *invalid* on some other calculators and big computers! Machines do exist which calculate $\ln(u)$ and/or $1 - u$ with small *absolute* error, but large *relative* error when u is near 1; on those machines the foregoing calculations must be wrong or much more complicated, often both. (Refer to the discussion under Level 2 for more about this.)

Back to Susan's sum. By using the foregoing simple procedure to calculate $\lambda(i/n) = \ln(1 + i/n) = 3.567351591 \times 10^{-9}$, she obtains a better value:

$$(1 + i/n)^n = e^{n\,\lambda(i/n)} = 1.119072257$$

from which the correct total follows.

To understand the error in 3^{201}, note that this is calculated as $e^{201\ln(3)} = e^{220.821\ldots}$. To keep the final relative error below one unit in the 10th significant digit, 201 ln(3) would have to be calculated with an absolute error rather smaller than 10^{-10}, which would entail carrying at least 14 significant digits for that intermediate value. The calculator does carry 13 significant digits for certain intermediate calculations of its own, but a 14th digit would cost more than it's worth.

Level 1C: Complex Level 1

Most complex arithmetic functions cannot guarantee 9 or 10 correct significant digits in each of a result's real and imaginary parts separately, although the result will conform to the summary statement about functions in Level 1 provided f, F, and ϵ are interpreted as complex numbers. In other words, every complex function f in Level 1C will produce a calculated complex value $F = (1 + \epsilon)f$ whose small complex relative error ϵ must satisfy $|\epsilon| < 10^{-9}$. The complex functions in Level 1C are $\boxed{\times}$, $\boxed{\div}$, $\boxed{x^2}$, $\boxed{\text{LN}}$, $\boxed{\text{LOG}}$, $\boxed{\text{SIN}^{-1}}$, $\boxed{\text{COS}^{-1}}$, $\boxed{\text{TAN}^{-1}}$, $\boxed{\text{SINH}^{-1}}$, $\boxed{\text{COSH}^{-1}}$, and $\boxed{\text{TANH}^{-1}}$. Therefore, a function like $\lambda(z) = \ln(1 + z)$ can be calculated accurately for all z by the same program as given above and with the same explanation.

To understand why a complex result's real and imaginary parts might not individually be correct to 9 or 10 significant digits, consider $\boxed{\times}$, for example: $(a + ib) \times (c + id) = (ac - bd) + i(ad + bc)$ ideally. Try this with $a = c = 9.999999998$, $b = 9.999999999$, and $d = 9.999999997$; the exact value of the product's real part $(ac - bd)$ should then be

$$\begin{aligned}(9.999999998)^2 &- (9.999999999)(9.999999997)\\ &= 99.999999980000000004 - 99.999999980000000003\\ &= 10^{-18}\end{aligned}$$

which requires that at least 20 significant digits be carried during the intermediate calculation. The HP-15C carries 13 significant digits for internal intermediate results, and therefore obtains 0 instead of 10^{-18} for the real part, but this error is negligible compared to the imaginary part 199.9999999 .

Level 2: Correctly Rounded for Possibly Perturbed Input

Trigonometric Functions of Real Radian Angles

Recall example 3, which noted that the calculator's $\boxed{\pi}$ key delivers an approximation to π correct to 10 significant digits but still slightly different from π, so $0 = \sin(\pi) \neq \sin(\boxed{\pi})$ for which the calculator delivers

$$\boxed{\text{SIN}}(\boxed{\pi}) = -4.100000000 \times 10^{-10}.$$

This computed value is not quite the same as the true value

$$\sin(\boxed{\pi}) = -4.10206761537356... \times 10^{-10}.$$

Whether the discrepancy looks small (absolute error less than 2.1×10^{-13}) or relatively large (wrong in the fourth significant digit) for a 10-significant-digit calculator, the discrepancy deserves to be understood because it foreshadows other errors that look, at first sight, much more serious.

Consider

$$10^{14}\pi = 314159265358979.3238462643...$$

with $\sin(10^{14}\pi) = 0$ and

$$10^{14} \times \boxed{\pi} = 314159265400000$$

with $\boxed{\text{SIN}}(10^{14}\boxed{\pi}) = 0.7990550814$, although the true

$$\sin(10^{14}\boxed{\pi}) = -0.78387....$$

The wrong sign is an error too serious to ignore; it seems to suggest a defect in the calculator. To understand the error in trigonometric functions we must pay attention to small differences among π and two approximations to π:

true	$\pi = 3.1415926535897932384626433...$	
key	$\boxed{\pi} = 3.141592654$	(matches π to 10 digits)
internal	$p = 3.141592653590$	(matches π to 13 digits)

Then all is explained by the following formula for the calculated value: $\boxed{\text{SIN}}(x) = \sin(x\pi/p)$ to within ±0.6 units in its last (10th) significant digit.

More generally, if $\text{trig}(x)$ is any of the functions $\sin(x)$, $\cos(x)$, or $\tan(x)$, evaluated in real Radians mode, the HP-15C produces

$$\boxed{\text{TRIG}}(x) = \text{trig}(x\pi/p)$$

to within ±0.6 units in its 10th significant digit.

This formula has important practical implications:

- Since $\pi/p = 1 - 2.0676... \times 10^{-13}/p = 0.9999999999999342...$, the value produced by $\boxed{\text{TRIG}}(x)$ differs from $\text{trig}(x)$ by no more than can be attributed to two perturbations: one in the 10th significant digit of the output $\text{trig}(x)$, and one in the 13th significant digit of the input x.

  If x has been calculated and rounded to 10 significant digits, the error inherited in its 10th significant digit is probably orders of magnitude bigger than $\boxed{\text{TRIG}}$'s second perturbation in x's 13th significant digit, so this second perturbation can be ignored unless x is regarded as known or calculated exactly.

- Every trigonometric identity that does not explicitly involve π is satisfied to within roundoff in the 10th significant digit of the calculated values in the identity. For instance,

  $$\sin^2(x) + \cos^2(x) = 1, \text{ so } (\boxed{\text{SIN}}(x))^2 + (\boxed{\text{COS}}(x))^2 = 1$$

  $$\sin(x)/\cos(x) = \tan(x), \text{ so } \boxed{\text{SIN}}(x)/\boxed{\text{COS}}(x) = \boxed{\text{TAN}}(x)$$

  with each calculated result correct to nine significant digits for all x. Note that $\boxed{\text{COS}}(x)$ vanishes for no value of x representable exactly with just 10 significant digits. And if $2x$ can be calculated exactly given x,

  $$\sin(2x) = 2\sin(x)\cos(x), \text{ so } \boxed{\text{SIN}}(2x) = 2\boxed{\text{SIN}}(x)\boxed{\text{COS}}(x)$$

  to nine significant digits. Try the last identity for $x = 52174$ radians on the HP-15C:

  $$\boxed{\text{SIN}}(2x) = -0.00001100815000,$$

  $$2\boxed{\text{SIN}}(x)\boxed{\text{COS}}(x) = -0.00001100815000\,.$$

  Note the close agreement even though for this x, $\sin(2x) = 2\sin(x)\cos(x) = -0.0000110150176...$ disagrees with $\boxed{\text{SIN}}(2x)$ in its fourth significant digit. The same identities are satisfied by $\boxed{\text{TRIG}}(x)$ values as by $\text{trig}(x)$ values even though $\boxed{\text{TRIG}}(x)$ and $\text{trig}(x)$ may disagree.

- Despite the two kinds of errors in $\boxed{\text{TRIG}}$, its computed values preserve familiar relationships wherever possible:

  - Sign symmetry: $\boxed{\text{COS}}(-x) = \boxed{\text{COS}}(x)$
$\boxed{\text{SIN}}(-x) = -\boxed{\text{SIN}}(x)$

- Monotonicity: if $\mathrm{trig}(x) \geq \mathrm{trig}(y)$,
  then $\boxed{\mathrm{TRIG}}(x) \geq \boxed{\mathrm{TRIG}}(y)$
  (provided $|x - y| < 3$)
- Limiting inequalities: $\boxed{\mathrm{SIN}}(x)/x \leq 1$ for all $x \neq 0$
  $\boxed{\mathrm{TAN}}(x)/x \geq 1$ for $0 < |x| < \pi/2$
  $-1 \leq \boxed{\mathrm{SIN}}(x)$ and $\boxed{\mathrm{COS}}(x) \leq 1$
  for all x

What do these properties imply for engineering calculations? *You don't have to remember them!*

In general, engineering calculations will not be affected by the difference between p and π, because the consequences of that difference in the formula defining $\boxed{\mathrm{TRIG}}(x)$ above are swamped by the difference between $\boxed{\pi}$ and π and by ordinary unavoidable roundoff in x or in $\mathrm{trig}(x)$. For engineering purposes, the ratio $\pi/p = 0.9999999999999342...$ could be replaced by 1 without visible effect upon the behavior of $\boxed{\mathrm{TRIG}}$.

Example 5: Lunar Phases. If the distance between our Earth and its moon were known accurately, we could calculate the phase difference between radar signals transmitted to and reflected from the moon. In this calculation the phase shift introduced by $p \neq \pi$ has less effect than changing the distance between Earth and moon by as little as the thickness of this page. Moreover, the calculation of the strength, direction, and rate of change of radiated signals near the moon or reflected signals near the Earth, calculations that depend upon the trigonometric identities' continuing validity, are unaffected by the fact that $p \neq \pi$; they rely instead upon the fact that p is a constant (independent of x in the formula for $\boxed{\mathrm{TRIG}}(x)$), and that constant is very near π.

The HP-15C's keyboard functions that involve p are the trigonometric functions $\boxed{\mathrm{SIN}}$, $\boxed{\mathrm{COS}}$, and $\boxed{\mathrm{TAN}}$ for real and complex arguments; hyperbolic functions $\boxed{\mathrm{SINH}}$, $\boxed{\mathrm{COSH}}$, and $\boxed{\mathrm{TANH}}$ for complex arguments; complex operations $\boxed{e^x}$, $\boxed{10^x}$, and $\boxed{y^x}$; and real and complex $\boxed{\rightarrow\mathrm{R}}$.

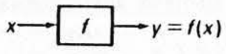
It all seems like much ado about very little. After a blizzard of formulas and examples, we conclude that the error caused by $p \neq \pi$ is negligible for engineering purposes, so we need not have bothered to know about it. That is the burden that conscientious error analysts must bear; if they merely took for granted that small errors are negligible, they might be wrong.

Backward Error Analysis

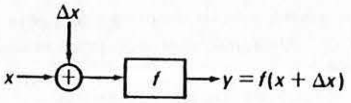
Until the late 1950's, most computer experts inclined to paranoia in their assessments of the damage done to numerical computations by rounding errors. To justify their paranoia, they could cite published error analyses like the one from which a famous scientist concluded that matrices as large as 40 × 40 were almost certainly impossible to invert numerically in the face of roundoff. However, by the mid 1960's matrices as large as 100 × 100 were being inverted routinely, and nowadays equations with hundreds of thousands of unknowns are being solved during geodetic calculations worldwide. How can we reconcile these accomplishments with the fact that that famous scientist's mathematical analysis was quite correct?

We understand better now than then why different formulas to calculate the same result might differ utterly in their degradation by rounding errors. For instance, we understand why the normal equations belonging to certain least-squares problems can be solved only in arithmetic carrying extravagantly high precision; this is what that famous scientist actually proved. We also know new procedures (one is presented on page 140) that can solve the same least-squares problems without carrying much more precision than suffices to represent the data. The new and better numerical procedures are not obvious, and might never have been found but for new and better techniques of error analysis by which we have learned to distinguish formulas that are hypersensitive to rounding errors from formulas that aren't. One of the new (in 1957) techniques is now called "backward error analysis," and you have already seen it in action twice: first, it explained why the procedure that calculates $\lambda(x)$ is accurate enough to dispel the inaccuracy in example 2; next, it explained why the calculator's $\boxed{\mathrm{TRIG}}$ functions very nearly satisfy the same identities as are satisfied by trig functions even for huge radian arguments x at which $\boxed{\mathrm{TRIG}}(x)$ and $\mathrm{trig}(x)$ can be very different. The following paragraphs explain backward error analysis itself in general terms.

Consider some system F intended to transform an input x into an output $y = f(x)$. For instance, F could be a signal amplifier, a filter, a transducer, a control system, a refinery, a country's economy, a computer program, or a calculator. The input and output need not be numbers; they could be sets of numbers or matrices or anything else quantitative. Were the input x to be contaminated by noise Δx, then in consequence the output $y + \Delta y = f(x + \Delta x)$ would generally be contaminated by noise $\Delta y = f(x + \Delta x) - f(x)$.



No Noise



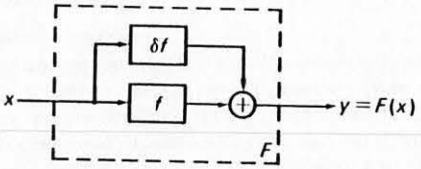
Noisy Input

Some transformations f are stable in the presence of input noise; they keep Δy relatively small as long as Δx is relatively small. Other transformations f may be unstable in the presence of noise because certain relatively small input noises Δx cause relatively huge perturbations Δy in the output. In general, the input noise Δx will be colored in some way by the intended transformation f on the way from input to output noise Δy, and no diminution in Δy is possible without either diminishing Δx or changing f. Having accepted f as a specification for performance or as a goal for design, we must acquiesce to the way f colors noise at its input.

The real system F differs from the intended f because of noise or other discrepancies inside F. Before we can appraise the consequences of that internal noise we must find a way to represent it, a notation. The simplest way is to write

$$F(x) = (f + \delta f)(x)$$

where the perturbation δf represents the internal noise in F.



One Small Output Perturbation (Level 1)

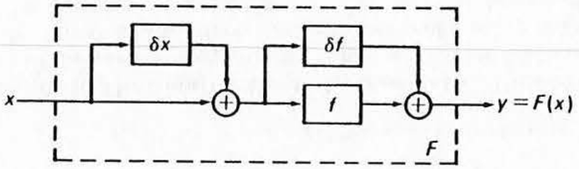
We hope the noise term δf is negligible compared with f. When that hope is fulfilled, we classify F in Level 1 for the purposes of exposition; this means that the noise internal to F can be explained as *one* small addition δf to the intended output f.

For example, $F(x) = \boxed{\text{LN}}(x)$ is classified in Level 1 because the dozens of small errors committed by the HP-15C during its calculation of $F(x) = (f + \delta f)(x)$ amounts to a perturbation $\delta f(x)$ smaller than 0.6 in the last (10th) significant digit of the desired output $f(x) = \ln(x)$. But $F(x) = \boxed{\text{SIN}}(x)$ is not in Level 1 for radian x because $F(x)$ can differ too much from $f(x) = \sin(x)$; for instance $F(10^{14}\boxed{\pi}) = 0.799...$ is opposite in sign from $f(10^{14}\boxed{\pi}) = -0.784...$, so the equation $F(x) = (f + \delta f)(x)$ can be true only if δf is sometimes rather bigger than f, which looks bad.

Real systems more often resemble $\boxed{\text{SIN}}$ than $\boxed{\text{LN}}$. Noise in most real systems can accumulate occasionally to swamp the desired output, at least for some inputs, and yet such systems do not necessarily deserve condemnation. Many a real system F operates reliably because its internal noise, though sometimes large, never causes appreciably more harm than might be caused by some tolerably small perturbation δx to the input signal x. Such systems can be represented as

$$F(x) = (f + \delta f)(x + \delta x)$$

where δf is always small compared with f and δx is always smaller than or comparable with the noise Δx expected to contaminate x. The two noise terms δf and δx are hypothetical noises introduced to explain diverse noise sources actually distributed throughout F. Some of the noise appears as a tolerably small perturbation δx to the input—hence the term "backward error analysis." Such a system F, whose noise can be accounted for by two tolerably small perturbations, is therefore classified into Level 2 for purposes of exposition.



Small Input and Output Perturbations (Level 2)

No difference will be perceived at first between Level 1 and Level 2 by readers accustomed to linear systems and small signals because such systems' errors can be referred indiscriminately to output or input. However, other more general systems that are digital or nonlinear do not admit arbitrary reattribution of output noise to input noise nor vice-versa.

For example, can all the error in $\boxed{\text{COS}}$ be attributed, merely by writing $\boxed{\text{COS}}(x) = \cos(x + \delta x)$, to an input perturbation δx small compared with the input x? Not when x is very small. For instance, when x approaches 10^{-5} radians, then $\cos(x)$ falls very near 0.99999999995 and must then round to either $1 = \cos(0)$ or $0.9999999999 = \cos(1.414... \times 10^{-5})$. Therefore $\boxed{\text{COS}}(x) = \cos(x + \delta x)$ is true only if δx is allowed to be relatively large, nearly as large as x when x is very small. If we wish to explain the error in $\boxed{\text{COS}}$ by using only relatively small perturbations, we need at least two of them: one a perturbation $\delta x = (-6.58... \times 10^{-14})x$ smaller than roundoff in the input; and another in the output comparable with roundoff there, so that $\boxed{\text{COS}}(x) = (\cos + \delta\cos)(x + \delta x)$ for some unknown $|\delta\cos| \leqslant (6 \times 10^{-10})|\cos|$.

Like $\boxed{\text{COS}}$, every system F in Level 2 is characterized by just two small tolerances—call them ϵ and η—that sum up all you have to know about that system's internal noise. The tolerance ϵ constrains a hypothetical output noise, $|\delta f| \leqslant \epsilon|f|$, and η constrains a hypothetical input noise, $|\delta x| \leqslant \eta|x|$, that might appear in a simple formula like

$$F(x) = (f + \delta f)(x + \delta x) \qquad \text{for } |\delta f| \leqslant \epsilon|f| \quad \text{and} \quad |\delta x| \leqslant \eta|x|.$$

The goal of backward error analysis is to ascertain that all the internal noise of F really can be encompassed by so simple a formula with satisfactorily small tolerances ϵ and η. At its best, backward error analysis confirms that *the realized value $F(x)$ scarcely differs from the ideal value $f(x + \delta x)$ that would have been produced by an input $x + \delta x$ scarcely different from the actual input x*, and gives the word "scarcely" a quantitative meaning (ϵ and η). But, backward error analysis succeeds only for systems F designed very carefully to ensure that every internal noise source is equivalent at worst to a tolerably small input or output perturbation. First attempts at system design, especially programs to perform numerical computations, often suffer from internal noise in a more complicated and disagreeable way illustrated by the following example.

Example 6: The Smaller Root of a Quadratic. The two roots x and y of the quadratic equation $c - 2bz + az^2 = 0$ are real whenever $d = b^2 - ac$ is nonnegative. Then the root y of smaller magnitude can be regarded as a function $y = f(a,b,c)$ of the quadratic's coefficients

$$f(a,b,c) = \begin{cases} (b - \sqrt{d}\,\mathrm{sgn}(b))/a & \text{if } a \neq 0 \\ (c/b)/2 & \text{otherwise.} \end{cases}$$

Were this formula translated directly in a program $F(a, b, c)$ intended to calculate $f(a, b, c)$, then whenever ac is so small compared with b^2 that the computed value of d rounds to b^2, that program could deliver $F = 0$ even though $f \neq 0$. So drastic an error cannot be explained by backward error analysis because no relatively small perturbations to each coefficient a, b, and c could drive c to zero, as would be necessary to change the smaller root y into 0. On the other hand, the algebraically equivalent formula

$$f(a,b,c) = \begin{cases} c/(b + \sqrt{d}\,\mathrm{sgn}(b)) & \text{if divisor is nonzero} \\ 0 & \text{otherwise} \end{cases}$$

translates into a much more accurate program F whose errors do no more damage than would a perturbation in the last (10th) significant digit of c. Such a program will be listed later (page 205) and must be used in those instances, common in engineering, when the smaller root y is needed accurately despite the fact that the quadratic's other unwanted root is relatively large.

Almost all the functions built into the HP-15C have been designed so that backward error analysis will account for their errors satisfactorily. The exceptions are $\boxed{\text{SOLVE}}$, $\boxed{\int_x^y}$, and the statistics keys $\boxed{\text{s}}$, $\boxed{\text{L.R.}}$, and $\boxed{\hat{y},\text{r}}$ which can malfunction in certain pathological cases. Otherwise, every calculator function F intended to produce $f(x)$ produces instead a value $F(x)$ no farther from $f(x)$ than if first x had been perturbed to $x + \delta x$ with $|\delta x| \leqslant \eta|x|$, then $f(x + \delta x)$ were perturbed to $(f + \delta f)(x + \delta x)$ with $|\delta f| \leqslant \epsilon|f|$. The tolerances η and ϵ vary a little from function to function; roughly speaking,

$\eta = 0$ and $\epsilon < 10^{-9}$	for all functions in Level 1,
$\eta < 10^{-12}$ and $\epsilon < 6 \times 10^{-10}$	for other real and complex functions.

For matrix operations, the magnitudes $|\delta x|$, $|x|$, $|\delta f|$, and $|f|$ must be replaced by matrix norms $\|\delta \mathbf{x}\|$, $\|\mathbf{x}\|$, $\|\delta \mathbf{f}\|$, and $\|\mathbf{f}\|$ respectively, which are explained in section 4 and evaluated using $\boxed{\text{MATRIX}}$ 7 or $\boxed{\text{MATRIX}}$ 8. Then all matrix functions not in Level 1 fall into Level 2 with roughly

$\eta \leqslant 10^{-12}n$ and $\epsilon < 10^{-9}$	for matrix operations (other than determinant $\boxed{\text{MATRIX}}$ 9, $\boxed{\div}$, and $\boxed{1/x}$)
$\eta \leqslant 10^{-9}n$ and $\epsilon < 10^{-9}$	for determinant $\boxed{\text{MATRIX}}$ 9, $\boxed{1/x}$, and $\boxed{\div}$ with a matrix divisor

where n is the largest dimension of any matrix involved in the operation.

The implications of successful backward error analysis look simple only when the input data x comes contaminated by unavoidable and uncorrelated noise Δx, as is often the case. Then when we wish to calculate $f(x)$, the best we could hope to get is $f(x + \Delta x)$, but we actually get $F(x + \Delta x) = (f + \delta f)(x + \Delta x + \delta x)$, where $|\delta f| \leqslant \epsilon|f|$ and $|\delta x| \leqslant \eta|x|$.

What we get is scarcely worse than the best we could hope for provided the tolerances ϵ and η are small enough, particularly if $|\Delta x|$ is likely to be at least roughly as big as $\eta|x|$. Of course, the best we could hope for may be very bad, especially if f possesses a singularity closer to x than the tolerances upon x's perturbations Δx and δx.

Backward Error Analysis Versus Singularities

The word "singularity" refers to both a special value of the argument x and to the way $f(x)$ misbehaves as x approaches that special value. Most commonly, $f(x)$ or its first derivative $f'(x)$ may become infinite or violently oscillatory as x approaches the singularity. Sometimes the singularities of $\ln|f|$ are called singularities of f, thereby including the zeros of f among its singularities; this makes sense when the relative accuracy of a computation of f is at issue, as we shall see. For our purposes the meaning of "singularity" can be left a little vague.

What we usually want to do with singularities is avoid or neutralize them. For instance, the function

$$c(x) = \begin{cases} (1 - \cos x)/x^2 & \text{if } x \neq 0 \\ 1/2 & \text{otherwise} \end{cases}$$

has no singularity at $x = 0$ even though its constituents $1 - \cos x$ and x^2 (actually, their logarithms) do behave singularly as x approaches 0. The constituent singularities cause trouble for the program that calculates $c(x)$. Most of the trouble is neutralized by the choice of a better formula

$$c(x) = \begin{cases} \dfrac{1}{2}\left(\dfrac{\sin(x/2)}{x/2}\right)^2 & \text{if } x/2 \neq 0 \\ 1/2 & \text{otherwise.} \end{cases}$$

Now the singularity can be avoided entirely by testing whether $x/2 = 0$ in the program that calculates $c(x)$.

Backward error analysis complicates singularities in a way that is easiest to illustrate with the function $\lambda(x) = \ln(1 + x)$ that solved the savings problem in example 2. The procedure used there calculated $u = 1 + x$ (rounded) $= 1 + x + \Delta x$. Then

$$\lambda(x) = \begin{cases} x & \text{if } u = 1 \\ \ln(u)\,x/(u - 1) & \text{otherwise.} \end{cases}$$

This procedure exploits the fact that $\lambda(x)/x$ has a removable singularity at $x = 0$, which means that $\lambda(x)/x$ varies continuously and approaches 1 as x approaches 0. Therefore, $\lambda(x)/x$ is relatively closely approximated by $\lambda(x + \Delta x)/(x + \Delta x)$ when $|\Delta x| < 10^{-9}$, and hence

$$\lambda(x) = x(\lambda(x)/x) \approx x(\lambda(x + \Delta x)/(x + \Delta x)) = x(\ln(u)/(u - 1)),$$

all calculated accurately because $\boxed{\text{LN}}$ is in Level 1. What might happen if $\boxed{\text{LN}}$ were in Level 2 instead?

If $\boxed{\text{LN}}$ were in Level 2, then "successful" backward error analysis would show that, for arguments u near 1, $\boxed{\text{LN}}(u) = \ln(u + \delta u)$ with $|\delta u| < 10^{-9}$. Then the procedure above would produce not $x(\ln(u)/(u - 1))$, but

$$\begin{aligned} x(\ln(u + \delta u)/(u - 1)) &= x\lambda(x + \Delta x + \delta u)/(x + \Delta x) \\ &= x(\lambda(x + \Delta x + \delta u)/(x + \Delta x + \delta u))\frac{x + \Delta x + \delta u}{x + \Delta x} \\ &\approx x(\lambda(x)/x)(1 + \delta u/(x + \Delta x)) \\ &= \lambda(x)(1 + \delta u/(x + \Delta x)). \end{aligned}$$

When $|x + \Delta x|$ is not much bigger than 10^{-9}, the last expression can be utterly different from $\lambda(x)$. Therefore, the procedure that solved example 2 would fail on machines whose $\boxed{\text{LN}}$ is not in Level 1. There *are* such machines, and on them the procedure does collapse for certain otherwise innocuous inputs. Similar failures also occur on machines that produce $(u + \delta' u) - 1$ instead of $u - 1$ because their $\boxed{-}$ function lies in Level 2 instead of Level 1. And those machines that produce $\ln(u + \delta u)/(u + \delta' u - 1)$ instead of $\ln(u)/(u-1)$, because both $\boxed{\text{LN}}$ and $\boxed{-}$ lie in Level 2, would be doubly vulnerable but for an ill-understood accident that usually correlates the two backward errors δu and $\delta' u$ in such a way as causes only half the significant digits of the computed λ, instead of all of them, to be wrong.

Summary to Here

Now that the complexity injected by backward error analysis into singularities has been exposed, the time has come to summarize, to simplify, and to consolidate what has been discussed so far.

- Many numerical procedures produce results too wrong to be justified by any satisfactory error analysis, backward or not.
- Some numerical procedures produce results only slightly worse than would have been obtained by exactly solving a problem differing only slightly from the given problem. Such procedures, classified in Level 2 for our purposes, are widely accepted as satisfactory from the point of view of backward error analysis.
- Procedures in Level 2 can produce results relatively far from what would have been obtained had no errors at all been committed, but large errors can result only for data relatively near a singularity of the function being computed.
- Procedures in Level 1 produce relatively accurate results regardless of near approach to a singularity. Such procedures are rare, but preferable if only because their results are easier to interpret, especially when several variables are involved.

A simple example illustrates all four points.

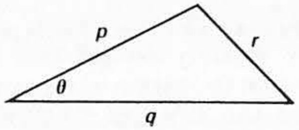
Example 7: The Angle in a Triangle. The cosine law for triangles says

$$r^2 = p^2 + q^2 - 2pq\cos\theta$$

for the figure shown below. Engineering and scientific calculations often require that the angle θ be calculated from given values p, q, and r for the length of the triangle's sides. This calculation is feasible provided $0 < p \leqslant q + r$, $0 < q \leqslant p + r$, and $0 \leqslant r \leqslant p + q$, and then

$$0 \leqslant \theta = \cos^{-1}(((p^2 + q^2) - r^2)/(2pq)) \leqslant 180°;$$

otherwise, no triangle exists with those side lengths, or else $\theta = 0/0$ is indeterminate.



The foregoing formula for θ defines a function $\theta = f(p,q,r)$ and also in a natural way, a program $F(p,q,r)$ intended to calculate the function. That program is labeled "A" below, with results $F_A(p,q,r)$ tabulated for certain inputs p, q, and r corresponding to sliver-shaped triangles for which the formula suffers badly from roundoff. The numerical unreliability of this formula is well known as is that of the algebraically equivalent but more reliable formula $\theta = f(p,q,r) = 2\tan^{-1}\sqrt{ab/(cs)}$, where $s = (p + q + r)/2$, $a = s - p$, $b = s - q$, and $c = s - r$. Another program $F(p,q,r)$ based upon this better formula is labeled "B" below, with results $F_B(p,q,r)$ for selected inputs. Apparently F_B is not much more reliable than F_A. Most of the poor results could be explained by backward error analysis if we assume that the calculations yield $F(p,q,r) = f(p + \delta p, q + \delta q, r + \delta r)$ for unknown but small perturbations satisfying $|\delta p| < 10^{-9}|p|$, etc. Even if this explanation were true, it would have perplexing and disagreeable consequences, because the angles in sliver-shaped triangles can change relatively drastically when the sides are perturbed relatively slightly; $f(p,q,r)$ is relatively unstable for marginal inputs.

Actually the preceding explanation is false. No backward error analysis could account for the results tabulated for F_A and F_B under case 1 below unless perturbations δp, δq, and δr were allowed to corrupt the fifth significant digit of the input, changing 1 to 1.0001 or 0.9999. That much is too much noise to tolerate in a 10-digit calculation. A better program by far is F_C, labeled "C" and explained shortly afterwards.

The three bottom lines in the table below show results for three programs "A", "B", and "C" based upon three different formulas $F(p,q,r)$ all algebraically equivalent to

$$\theta = f(p,q,r) = \cos^{-1}((p^2+q^2-r^2)/(2pq)).$$

Disparate Results from Three Programs F_A, F_B, F_C

	Case 1	Case 2	Case 3
p	1.	9.999999996	10.
q	1.	9.999999994	5.000000001
r	1.00005×10^{-5}	3×10^{-9}	15.
F_A	0.	0.	180.
F_B	5.73072×10^{-4}	Error 0	180.
F_C	5.72986×10^{-4}	1.28117×10^{-8}	179.9985965
	Case 4	**Case 5**	**Case 6**
p	0.527864055	9.999999996	9.999999999
q	9.472135941	3×10^{-9}	9.999999999
r	9.999999996	9.999999994	20.
F_A	Error 0	48.18968509	180.
F_B	Error 0	Error 0	180.
F_C	180.	48.18968510	Error 0
	Case 7	**Case 8**	**Case 9**
p	1.00002	3.162277662	3.162277662
q	1.00002	2.3×10^{-9}	1.5555×10^{-6}
r	2.00004	3.162277661	3.162277661
F_A	Error 0	90.	90.
F_B	180.	70.52877936	89.96318706
F_C	180.	64.22853822	89.96315156

To use a program, key in p ENTER q ENTER r, run program "A", "B", or "C", and wait to see the program's approximation F to $\theta = f$. Only program "C" is reliable.

```
Keystrokes          Display

g DEG
g P/R
f CLEAR PRGM        000-
f LBL A             001-42,21,11
g x²                002-    43 11
x≷y                 003-       34
g x²                004-    43 11
g LSTx              005-    43 36
g R↑                006-    43 33
×                   007-       20
x≷y                 008-       34
g LSTx              009-    43 36
g x²                010-    43 11
+                   011-       40
g R↑                012-    43 33
-                   013-       30
x≷y                 014-       34
ENTER               015-       36
+                   016-       40
÷                   017-       10
g COS⁻¹             018-    43 24
g RTN               019-    43 32
f LBL B             020-42,21,12
STO 1               021-    44  1
ENTER               022-       36
g R↑                023-    43 33
STO + 1             024-44,40, 1
g R↑                025-    43 33
STO + 1             026-44,40, 1
2                   027-        2
STO ÷ 1             028-44,10, 1
R↓                  029-       33
RCL - 1             030-45,30, 1
x≷y                 031-       34
RCL - 1             032-45,30, 1
×                   033-       20
√x                  034-       11
x≷y                 035-       34
RCL - 1             036-45,30, 1
RCL × 1             037-45,20, 1
```

Keystrokes	Display
CHS	038- 16
√x	039- 11
g →P	040- 43 1
R↓	041- 33
×	042- 20
g RTN	043- 43 32
f LBL C	044-42,21,13
STO 0	045- 44 0
R↓	046- 33
g x≤y	047- 43 10
x≷y	048- 34
STO 1	049- 44 1
STO + 0	050-44,40, 0
x≷y	051- 34
STO + 0	052-44,40, 0
−	053- 30
g R↑	054- 43 33
STO − 1	055-44,30, 1
g LSTx	056- 43 36
ENTER	057- 36
RCL + 1	058-45,40, 1
√x	059- 11
f x≷ 0	060-42, 4, 0
√x	061- 11
STO × 0	062-44,20, 0
g CLx	063- 43 35
+	064- 40
R↓	065- 33
+	066- 40
f x≷ 1	067-42, 4, 1
g R↑	068- 43 33
g LSTx	069- 43 36
g x≤y	070- 43 10
GTO .9	071- 22 .9
R↓	072- 33
g TEST 2	073-43,30, 2
√x	074- 11
x≷y	075- 34
GTO .8	076- 22 .8
f LBL .9	077-42,21, .9

Keystrokes	Display
g TEST 2	078-43,30, 2
√x	079- 11
g R↑	080- 43 33
f LBL .8	081-42,21, .8
−	082- 30
√x	083- 11
RCL 1	084- 45 1
√x	085- 11
×	086- 20
RCL 0	087- 45 0
g →P	088- 43 1
g x=0	089- 43 20
÷	090- 10
x≷y	091- 34
ENTER	092- 36
+	093- 40
g RTN	094- 43 32
g P/R	

The results $F_C(p,q,r)$ are correct to at least nine significant digits. They are obtained from a program "C" that is utterly reliable though rather longer than the unreliable programs "A" and "B". The method underlying program "C" is:

1. If $p < q$, then swap them to ensure $p \geqslant q$.
2. Calculate $b = (p - q) + r$, $c = (p - r) + q$, and $s = (p + r) + q$.
3. Calculate

$$a = \begin{cases} r - (p - q) & \text{if } q \geqslant r \geqslant 0 \\ q - (p - r) & \text{if } r > q \geqslant 0 \\ \textbf{Error 0} & \text{otherwise (no triangle exists).} \end{cases}$$

4. Calculate $F_C(p,q,r) = 2\tan^{-1}(\sqrt{ab}/\sqrt{cs})$.

This procedure delivers $F_C(p,q,r) = \theta$ correct to almost nine significant digits, a result surely easier to use and interpret than the results given by the other better-known formulas. But this procedure's internal workings are hard to explain; indeed, the procedure may malfunction on some calculators and computers.

The procedure works impeccably on only certain machines like the HP-15C, whose subtraction operation is free from avoidable error and therefore enjoys the following property: Whenever y lies between $x/2$ and $2x$, the subtraction operation introduces no roundoff error into the calculated value of $x - y$. Consequently, whenever cancellation might leave relatively large errors contaminating a, b, or c, the pertinent difference $(p - q)$ or $(p - r)$ turns out to be free from error, and then cancellation turns out to be advantageous!

Cancellation remains troublesome on those other machines that calculate $(x + \delta x) - (y + \delta y)$ instead of $x - y$ even though neither δx nor δy amounts to as much as one unit in the last significant digit carried in x or y respectively. Those machines deliver $F_C(p,q,r) = f(p + \delta p, q + \delta q, r + \delta r)$ with end-figure perturbations δp, δq, and δr that always seem negligible from the viewpoint of backward error analysis, but which can have disconcerting consequences. For instance, only one of the triples (p,q,r) or $(p + \delta p, q + \delta q, r + \delta r)$, not both, might constitute the edge lengths of a feasible triangle, so F_C might produce an error message when it shouldn't, or vice-versa, on those machines.

Backward Error Analysis of Matrix Inversion

The usual measure of the magnitude of a matrix $\mathbf{X}$ is a norm $\|\mathbf{X}\|$ such as is calculated by either $\boxed{\text{MATRIX}}$ 7 or $\boxed{\text{MATRIX}}$ 8; we shall use the former norm, the row norm

$$\|\mathbf{X}\| = \max_i \sum_j |x_{ij}|$$

in what follows. This norm has properties similar to those of the length of a vector and also the multiplicative property

$$\|\mathbf{XY}\| \leqslant \|\mathbf{X}\|\,\|\mathbf{Y}\|.$$

When the equation $\mathbf{Ax} = \mathbf{b}$ is solved numerically with a given $n \times n$ matrix $\mathbf{A}$ and column vector $\mathbf{b}$, the calculated solution is a column vector $\mathbf{c}$ which satisfies nearly the same equation as does $\mathbf{x}$, namely

$$(\mathbf{A} + \delta\mathbf{A})\mathbf{c} = \mathbf{b}$$

with $\|\delta\mathbf{A}\| < 10^{-9}n\,\|\mathbf{A}\|$.

Consequently the residual $\mathbf{b} - \mathbf{Ac} = (\delta\mathbf{A})\mathbf{c}$ is always relatively small; quite often the residual norm $\|\mathbf{b} - \mathbf{Ac}\|$ is smaller than $\|\mathbf{b} - \mathbf{A\bar{x}}\|$ where $\mathbf{\bar{x}}$ is obtained from the true solution $\mathbf{x}$ by rounding each of its elements to 10 significant digits. Consequently, $\mathbf{c}$ can differ significantly from $\mathbf{x}$ only if $\mathbf{A}$ is nearly singular, or equivalently only if $\|\mathbf{A}^{-1}\|$ is relatively large compared with $1/\|\mathbf{A}\|$;

$$\begin{aligned} \|\mathbf{x} - \mathbf{c}\| &= \|\mathbf{A}^{-1}(\mathbf{b} - \mathbf{Ac})\| \\ &\leqslant \|\mathbf{A}^{-1}\|\,\|\delta\mathbf{A}\|\,\|\mathbf{c}\| \\ &\leqslant 10^{-9}n\,\|\mathbf{c}\|/\sigma(\mathbf{A}) \end{aligned}$$

where $\sigma(\mathbf{A}) = 1/(\|\mathbf{A}\|\,\|\mathbf{A}^{-1}\|)$ is the reciprocal of the condition number and measures how relatively near to $\mathbf{A}$ is the nearest singular matrix $\mathbf{S}$, since

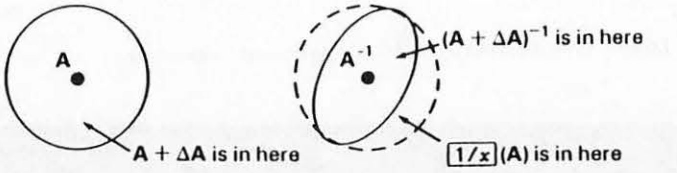
$$\min_{\det(\mathbf{S})=0} \|\mathbf{A} - \mathbf{S}\| = \sigma(\mathbf{A})\,\|\mathbf{A}\|.$$

These relations and some of their consequences are discussed extensively in section 4.

The calculation of $\mathbf{A}^{-1}$ is more complicated. Each column of the calculated inverse $\boxed{1/x}(\mathbf{A})$ is the corresponding column of some $(\mathbf{A} + \delta\mathbf{A})^{-1}$, but each column has its own small $\delta\mathbf{A}$. Consequently, no single small $\delta\mathbf{A}$, with $\|\delta\mathbf{A}\| \leqslant 10^{-9}n\,\|\mathbf{A}\|$, need exist satisfying

$$\|(\mathbf{A} + \delta\mathbf{A})^{-1} - \boxed{1/x}(\mathbf{A})\| \leqslant 10^{-9}\,\|\boxed{1/x}(\mathbf{A})\|$$

roughly. Usually such a $\delta\mathbf{A}$ exists, but not always. This does not violate the prior assertion that the matrix operations $\boxed{1/x}$ and $\boxed{\div}$ lie in Level 2; they are covered by the second assertion of the summary on page 194. The accuracy of $\boxed{1/x}(\mathbf{A})$ can be described in terms of the inverses of all matrices $\mathbf{A} + \Delta\mathbf{A}$ so near $\mathbf{A}$ that $\|\Delta\mathbf{A}\| \leqslant 10^{-9}n\|\mathbf{A}\|$; the worst among those $(\mathbf{A} + \Delta\mathbf{A})^{-1}$ is at least about as far from $\mathbf{A}^{-1}$ in norm as the calculated $\boxed{1/x}(\mathbf{A})$. The figure below illustrates the situation.



As $\mathbf{A} + \Delta\mathbf{A}$ runs through matrices with $\|\Delta\mathbf{A}\|$ at least about as large as roundoff in $\|\mathbf{A}\|$, its inverse $(\mathbf{A} + \Delta\mathbf{A})^{-1}$ must roam at least about as far from $\mathbf{A}^{-1}$ as the distance from $\mathbf{A}^{-1}$ to the computed $\boxed{1/x}(\mathbf{A})$. All these excursions are very small unless **A** is too near a singular matrix, in which case the matrix should be preconditioned away from near singularity. (Refer to section 4.)

If among those neighboring matrices $\mathbf{A} + \Delta\mathbf{A}$ lurk some that are singular, then many $(\mathbf{A} + \Delta\mathbf{A})^{-1}$ and $\boxed{1/x}(\mathbf{A})$ may differ utterly from $\mathbf{A}^{-1}$. However, the residual norm will always be relatively small:

$$\frac{\|\mathbf{A}(\mathbf{A} + \Delta\mathbf{A})^{-1} - \mathbf{I}\|}{\|\mathbf{A}\|\,\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|} \leqslant \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \leqslant 10^{-9}n.$$

This last inequality remains true when $\boxed{1/x}(\mathbf{A})$ replaces $(\mathbf{A} + \Delta\mathbf{A})^{-1}$.

If **A** is far enough from singularity that all

$$1/\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\| > 10^{-9}n\|\mathbf{A}\| \geqslant \|\Delta\mathbf{A}\|,$$

then also

$$\begin{aligned}\frac{\|\mathbf{A}^{-1} - (\mathbf{A} + \Delta\mathbf{A})^{-1}\|}{\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|} &\leqslant \frac{\|\Delta\mathbf{A}\|\,\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|}{1 - \|\Delta\mathbf{A}\|\,\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|}\\ &\leqslant \frac{10^{-9}n\|\mathbf{A}\|\,\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|}{1 - 10^{-9}n\|\mathbf{A}\|\,\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\|}.\end{aligned}$$

This inequality also remains true when $\boxed{1/x}(\mathbf{A})$ replaces $(\mathbf{A} + \Delta\mathbf{A})^{-1}$, and then everything on the right-hand side can be calculated, so the error in $\boxed{1/x}(\mathbf{A})$ cannot exceed a knowable amount. In other words, the radius of the dashed ball in the figure above can be calculated.

The estimates above tend to be pessimistic. However, to show why nothing much better is true in general, consider the matrix

$$\mathbf{X} = \begin{bmatrix} 0.00002 & -50{,}000 & 50{,}000.03 & -45 \\ 0 & 50{,}000 & -50{,}000.03 & 45 \\ 0 & 0 & 0.00002 & -50{,}000.03 \\ 0 & 0 & 0 & 52{,}000 \end{bmatrix}$$

and

$$\mathbf{X}^{-1} = \begin{bmatrix} 50{,}000 & 50{,}000 & p & q \\ 0 & 0.00002 & 50{,}000.03 & 48{,}076.98077\ldots \\ 0 & 0 & 50{,}000 & 48{,}076.95192\ldots \\ 0 & 0 & 0 & 0.00001923076923\ldots \end{bmatrix}.$$

Ideally, $p = q = 0$, but the HP-15C's approximation to $\mathbf{X}^{-1}$, namely $\boxed{1/x}(\mathbf{X})$, has $q = 9{,}643.269231$ instead, a relative error

$$\frac{\|\mathbf{X}^{-1} - \boxed{1/x}(\mathbf{X})\|}{\|\mathbf{X}^{-1}\|} = 0.0964\ldots,$$

nearly 10 percent. On the other hand, if $\mathbf{X} + \Delta\mathbf{X}$ differs from **X** only in its second column where -50,000 and 50,000 are replaced respectively by -50,000.000002 and 49,999.999998 (altered in the 11th significant digit), then $(\mathbf{X} + \Delta\mathbf{X})^{-1}$ differs significantly from $\mathbf{X}^{-1}$ only insofar as $p = 0$ and $q = 0$ must be replaced by $p = 10{,}000.00600\ldots$ and $q = 9{,}615.396154\ldots$. Hence,

$$\frac{\|\mathbf{X}^{-1} - (\mathbf{X} + \Delta\mathbf{X})^{-1}\|}{\|\mathbf{X}^{-1}\|} = 0.196\ldots;$$

the relative error in $(\mathbf{X} + \Delta\mathbf{X})^{-1}$ is nearly twice that in $\boxed{1/x}(\mathbf{X})$. Do not try to calculate $(\mathbf{X} + \Delta\mathbf{X})^{-1}$ directly, but use instead the formula

$$(\mathbf{X} - \mathbf{c}\mathbf{b}^T)^{-1} = \mathbf{X}^{-1} + \mathbf{X}^{-1}\mathbf{c}\mathbf{b}^T\mathbf{X}^{-1} / (1 - \mathbf{b}^T\mathbf{X}^{-1}\mathbf{c}),$$

which is valid for any column vector **c** and row vector $\mathbf{b}^T$, and specifically for

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ and } \mathbf{b}^T = [0 \quad 0.000002 \quad 0 \quad 0].$$

Despite that

$$\|\mathbf{X}^{-1} - \boxed{1/x}(\mathbf{X})\| < \|\mathbf{X}^{-1} - (\mathbf{X} + \Delta\mathbf{X})^{-1}\|,$$

it can be shown that no very small end-figure perturbation $\delta\mathbf{X}$ exists for which $(\mathbf{X} + \delta\mathbf{X})^{-1}$ matches $\boxed{1/x}(\mathbf{X})$ to more than five significant digits in norm.

Of course, none of these horrible things could happen if **X** were not so nearly singular. Because $\|\mathbf{X}\|\ \|\mathbf{X}^{-1}\| > 10^{10}$, a change in **X** amounting to less than one unit in the 10th significant digit of $\|\mathbf{X}\|$ could make **X** singular; such a change might replace one of the diagonal elements 0.00002 of **X** by zero. Since **X** is so nearly singular, the accuracy of [1/x](**X**) in this case rather exceeds what might be expected in general. What makes this example special is bad scaling; **X** was obtained from an unexceptional matrix

$$\tilde{\mathbf{X}} = \begin{bmatrix} 2. & -5. & 5.000003 & -4.5\times 10^{-12} \\ 0 & 5. & -5.000003 & 4.5\times 10^{-12} \\ 0 & 0 & 2. & -5.000003 \\ 0 & 0 & 0 & 5.2 \end{bmatrix}$$

by multiplying each row and each column by a carefully chosen power of 10. Compensatory division of the columns and rows of the equally unexceptional matrix

$$\tilde{\mathbf{X}}^{-1} = \begin{bmatrix} 0.5 & 0.5 & p & q \\ 0 & 0.2 & 0.5000003 & 0.4807698077\ldots \\ 0 & 0 & 0.5 & 0.4807695192\ldots \\ 0 & 0 & 0 & 0.1923076923\ldots \end{bmatrix}$$

yielded $\mathbf{X}^{-1}$, with $p = q = 0$. The HP-15C calculates [1/x]($\tilde{\mathbf{X}}$) = $\tilde{\mathbf{X}}^{-1}$ except that $q = 0$ is replaced by $q = 9.6\times 10^{-11}$, a negligible change. This illustrates how drastically the perceived quality of computed results can be altered by scaling. (Refer to section 4 for more information about scaling.)

Is Backward Error Analysis a Good Idea?

The only good thing to be said for backward error analysis is that it explains internal errors in a way that liberates a system's user from having to know about internal details of the system. Given two tolerances, one upon the input noise δx and one upon the output noise δf, the user can analyze the consequences of internal noise in

$$F(x) = (f + \delta f)(x + \delta x)$$

by studying the noise propagation properties of the ideal system f without further reference to the possibly complex internal structure of F.

But backward error analysis is no panacea; it may explain errors but not excuse them. Because it complicates computations involving singularities, we have tried to eliminate the need for it wherever we could. If we knew how to eliminate the need for backward error analysis from every function built into the calculator, and to do so at tolerable cost, we would do that and simplify life for everyone. That simplicity would cost too much speed and memory for today's technology. The next example will illustrate the trade-offs involved.

Example 6 Continued. The program listed below solves the real quadratic equation $c - 2bz + az^2 = 0$ for real or complex roots.

To use the program, key the real constants into the stack (c [ENTER] b [ENTER] a) and run program "A".

The roots x and y will appear in the X- and Y-registers. If the roots are complex, the **C** annunciator turns on, indicating that Complex mode has been activated. The program uses labels "A" and ".9" and the Index register (but none of the other registers 0 to .9); therefore, the program may readily be called as a subroutine by other programs. The calling programs (after clearing flag 8 if necessary) can discover whether roots are real or complex by testing flag 8, which gets set only if roots are complex.

The roots x and y are so ordered that $|x| \geqslant |y|$ except possibly when $|x|$ and $|y|$ agree to more than nine significant digits. The roots are as accurate as if the coefficient c had first been perturbed in its 10th significant digit, the perturbed equation had been solved exactly, and its roots rounded to 10 significant digits. Consequently, the computed roots match the given quadratic's roots to at least five significant digits. More generally, if the roots x and y agree to n significant digits for some positive $n \leqslant 5$, then they are correct to at least $10 - n$ significant digits unless overflow or underflow occurs.

```
Keystrokes              Display

[g][P/R]
[f]CLEAR[PRGM]          000-
[f][LBL][A]             001-42,21,11
[ENTER]                 002-        36
[g][R↑]                 003-     43 33
[×]                     004-        20
[g][LSTx]               005-     43 36
```

Keystrokes	Display
x≷y	006- 34
g R↑	007- 43 33
STO I	008- 44 25
g x^2	009- 43 11
−	010- 30
g TEST 1	011-43,30, 1
GTO .9	012- 22 .9
CHS	013- 16
√x	014- 11
f x≷I	015-42, 4,25
g TEST 2	016-43,30, 2
RCL − I	017-45,30,25
g TEST 3	018-43,30, 3
RCL + I	019-45,40,25
g TEST 0	020-43,30, 0
÷	021- 10
g LSTx	022- 43 36
g R↑	023- 43 33
÷	024- 10
g RTN	025- 43 32
f LBL .9	026-42,21, .9
√x	027- 11
RCL I	028- 45 25
g R↑	029- 43 33
÷	030- 10
x≷y	031- 34
g LSTx	032- 43 36
÷	033- 10
f I	034- 42 25
ENTER	035- 36
f Re≷Im	036- 42 30
CHS	037- 16
f Re≷Im	038- 42 30
g RTN	039- 43 32
g P/R	

The method uses $d = b^2 - ac$.

If $d < 0$, then the roots are a complex conjugate pair

$$(b/a) \pm i\sqrt{-d}/a.$$

If $d \geqslant 0$, then the roots are real numbers x and y calculated by

$$s = b + \sqrt{d}\ \operatorname{sgn}(b)$$

$$x = s/a$$

$$y = \begin{cases} c/s & \text{if } s \neq 0 \\ 0 & \text{if } s = 0. \end{cases}$$

The s calculation avoids destructive cancellation.

When $a = 0 \neq b$, the larger root x, which should be ∞, encounters division by zero (**Error 0**) that can be cleared by pressing R↓ three times to exhibit the smaller root y correctly calculated. But when all three coefficients vanish, the **Error 0** message signals that both roots are arbitrary.

The results of several cases are summarized below.

	Case 1	Case 2	Case 3	Case 4
c	3	4	1	654,321
b	2	0	1	654,322
a	1	1	10^{-13}	654,323
Roots	Real	Complex	Real	Real
	3	$0 \pm 2i$	2×10^{13}	0.9999984717
	1		0.5	0.9999984717

	Case 5	Case 6
c	46,152,709	12,066,163
b	735,246	987,644
a	11,713	80,841
Roots	Real	Complex
	62.77179203	$12.21711755 \pm i0.001377461$
	62.77179203	

The last three cases show how severe are the results of perturbing the 10th significant digit of any coefficient of any quadratic whose roots are nearly coincident. The correct roots for these cases are

Case 4: 1 and 0.9999969434

Case 5: $62.77179203 \pm i8.5375 \times 10^{-5}$

Case 6: $12.21711755 \pm i0.001374514$

Despite errors in the fifth significant digit of the results, subroutine "A" suffices for almost all engineering and scientific applications of quadratic equations. Its results are correct to nine significant digits for most data, including c, b, and a representable exactly using only five significant digits; and the computed roots are correct to at least five significant digits in any case because they cannot be appreciably worse than if the data had been entered with errors in the 10th significant digit. Nonetheless, some readers will feel uneasy about results calculated to 10 significant digits but correct to only 5. If only to simplify their understanding of the relationship between input data and output results, they might still prefer roots correct to nine significant digits in all cases.

Programs do exist which, while carrying only 10 significant digits during arithmetic, will calculate the roots of any quadratic correctly to at least nine significant digits regardless of how nearly coincident those roots may be. All such programs calculate $d = b^2 - ac$ by some trick tantamount to carrying 20 significant digits whenever b^2 and ac nearly cancel, so those programs are a lot longer and slower than the simple subroutine "A" provided above. Subroutine "B" below, which uses such a trick,* is a very short program that guarantees nine correct significant digits on a 10-digit calculator. It uses labels "B", ".7", and ".8" and registers R_0 through R_9 and the Index register. To use it, key in c $\boxed{\text{ENTER}}$ b $\boxed{\text{ENTER}}$ a, run subroutine "B", and wait for results as before.

Keystrokes	Display
$\boxed{\text{g}}\boxed{\text{P/R}}$	
$\boxed{\text{f}}$ CLEAR $\boxed{\text{PRGM}}$	000–
$\boxed{\text{f}}\boxed{\text{LBL}}\boxed{\text{B}}$	001–42,21,12
$\boxed{\text{STO}}\boxed{\text{I}}$	002– 44 25
$\boxed{\text{R}\downarrow}$	003– 33
$\boxed{\text{STO}}$ 0	004– 44 0
$\boxed{\text{STO}}$ 8	005– 44 8
$\boxed{x \gtrless y}$	006– 34
$\boxed{\text{STO}}$ 1	007– 44 1
$\boxed{\text{STO}}$ 9	008– 44 9
$\boxed{\text{f}}\boxed{\text{SCI}}$ 2	009–42, 8, 2
$\boxed{\text{f}}\boxed{\text{LBL}}$ .8	010–42,21, .8
$\boxed{\text{f}}$ CLEAR $\boxed{\Sigma}$	011– 42 32
$\boxed{\text{RCL}}$ 8	012– 45 8
$\boxed{\text{STO}}$ 7	013– 44 7
$\boxed{\text{RCL}}\boxed{\div}\boxed{\text{I}}$	014–45,10,25
$\boxed{\text{g}}\boxed{\text{RND}}$	015– 43 34
$\boxed{\text{RCL}}\boxed{\text{I}}$	016– 45 25
$\boxed{\text{g}}\boxed{\Sigma-}$	017– 43 49
$\boxed{\text{RCL}}$ 9	018– 45 9
$\boxed{\text{f}}\boxed{x \gtrless}$ 7	019–42, 4, 7
$\boxed{x \gtrless y}$	020– 34
$\boxed{\text{RCL}}$ 8	021– 45 8
$\boxed{\text{g}}\boxed{\Sigma-}$	022– 43 49
$\boxed{\text{R}\downarrow}$	023– 33
$\boxed{\text{g}}\boxed{\Sigma-}$	024– 43 49
$\boxed{\text{RCL}}$ 7	025– 45 7
$\boxed{\text{g}}\boxed{\text{ABS}}$	026– 43 16
$\boxed{\text{RCL}}$ 9	027– 45 9
$\boxed{\text{g}}\boxed{\text{ABS}}$	028– 43 16
$\boxed{\text{g}}\boxed{x \leq y}$	029– 43 10
$\boxed{\text{GTO}}\boxed{\text{B}}$	030– 22 12
$\boxed{\text{ENTER}}$	031– 36
$\boxed{\text{g}}\boxed{\text{R}\uparrow}$	032– 43 33
$\boxed{\text{STO}}$ 8	033– 44 8
$\boxed{\text{RCL}}$ 7	034– 45 7
$\boxed{\text{STO}}$ 9	035– 44 9
$\boxed{\text{g}}\boxed{\text{ABS}}$	036– 43 16
$\boxed{\text{EEX}}$	037– 26
2	038– 2
0	039– 0
$\boxed{\times}$	040– 20
$\boxed{\text{RCL}}$ 1	041– 45 1
$\boxed{\text{g}}\boxed{\text{ABS}}$	042– 43 16
$\boxed{\text{g}}\boxed{x \leq y}$	043– 43 10

*Program "B" exploits a tricky property of the $\boxed{\Sigma-}$ and $\boxed{\Sigma+}$ keys whereby certain calculations can be carried out to 13 significant digits before being rounded back to 10.

Keystrokes	Display
[GTO] .8	044- 22 .8
[f] [LBL] [B]	045-42,21,12
[f] [FIX] 9	046-42, 7, 9
[RCL] 8	047- 45 8
[g] [x^2]	048- 43 11
[STO] 7	049- 44 7
[RCL] [I]	050- 45 25
[RCL] 9	051- 45 9
[g] [Σ-]	052- 43 49
[RCL] 7	053- 45 7
[g] [TEST] 2	054-43,30, 2
[GTO] .7	055- 22 .7
[√x]	056- 11
[f] [x≷] 0	057-42, 4, 0
[g] [TEST] 2	058-43,30, 2
[RCL] [-] 0	059-45,30, 0
[g] [TEST] 3	060-43,30, 3
[RCL] [+] 0	061-45,40, 0
[f] [x≷] 1	062-42, 4, 1
[g] [TEST] 0	063-43,30, 0
[RCL] [÷] 1	064-45,10, 1
[RCL] 1	065- 45 1
[RCL] [+] [I]	066-45,10,25
[g] [RTN]	067- 43 32
[f] [LBL] .7	068-42,21, .7
[CHS]	069- 16
[√x]	070- 11
[RCL] [÷] [I]	071-45,10,25
[ENTER]	072- 36
[CHS]	073- 16
[RCL] 0	074- 45 0
[RCL] [I]	075- 45 25
[÷]	076- 10
[x≷y]	077- 34
[f] [I]	078- 42 25
[ENTER]	079- 36
[g] [R↑]	080- 43 33
[f] [I]	081- 42 25
[g] [RTN]	082- 43 32
[g] [P/R]	

This program's accuracy is phenomenal: better than nine significant digits even for the imaginary parts of nearly indistinguishable complex roots (as when c = 4,877,163,849 and b = 4,877,262,613 and a = 4,877,361,379); if the roots are integers, real or complex, and if a = 1, then the roots are calculated exactly (as when $c = 1{,}219{,}332{,}937 \times 10^1$, b = 111,111.5, and a = 1). But the program is costly; it uses more than twice as much memory for both program and data as does subroutine "A", and much more time, to achieve nine significant digits of accuracy instead of five in a few cases that can hardly ever matter—simply because the quadratic's coefficients can hardly ever be calculated exactly. If any coefficient c, b, or a is uncertain by as much as one unit in its 10th significant digit, then subroutine "B" is overkill. Subroutine "B" is like Grandmother's expensive chinaware, reserved for special occasions, leaving subroutine "A" for everyday use.