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 $\|\mathbf{A}\|_R$:

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

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

$\|\mathbf{A}-\mathbf{B}\|_F = \sqrt{11} \approx 3.3$ (Frobenius norm),

$\|\mathbf{A}-\mathbf{B}\|_R = 3$ (row norm), and

$\|\mathbf{A}-\mathbf{B}\|_C = 4$ (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 $1 \leq K(\mathbf{A}) < \infty$ 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 $\mathbf{X}$ and $\mathbf{B}$ are nonzero vectors satisfying $\mathbf{AX} = \mathbf{B}$ for some square matrix $\mathbf{A}$. Suppose $\mathbf{A}$ 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 $\mathbf{A}$ 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 $\mathbf{X}$ and $\mathbf{B}$ are nonzero vectors satisfying $\mathbf{AX} = \mathbf{B}$ for some matrix $\mathbf{A}$. Suppose now that matrix $\mathbf{B}$ 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 $\mathbf{A}$ 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 $\mathbf{A}$ 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 $\mathbf{A}$ indicates that the matrix $\mathbf{A}$ 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 $\mathbf{S}$. That is, if $K(\mathbf{A})$ is large, then the relative difference between $\mathbf{A}$ and the closest singular matrix $\mathbf{S}$ is small. If the norm of $\mathbf{A}^{-1}$ is large, the difference between $\mathbf{A}$ and the closest singular matrix $\mathbf{S}$ 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

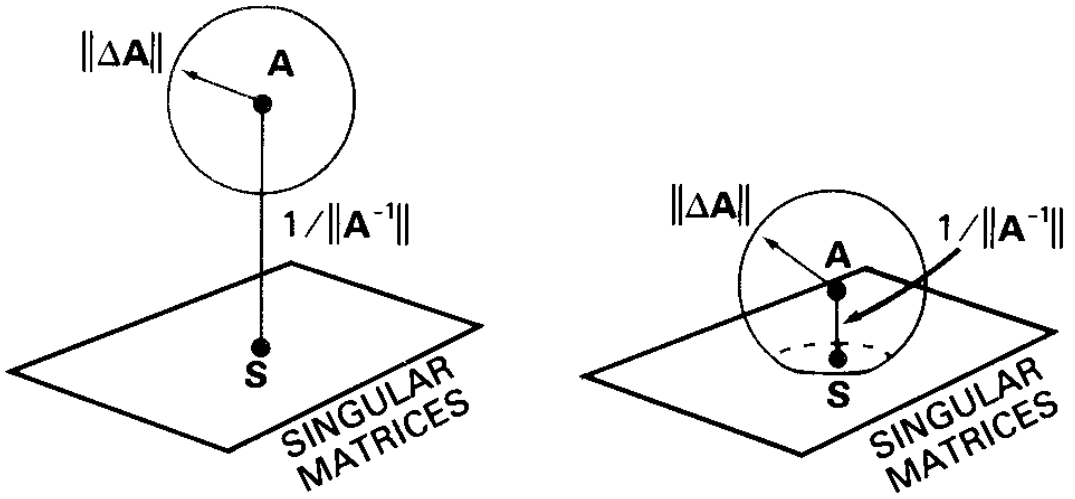
$$\Delta\mathbf{A} = \begin{bmatrix} 0 & -5 \times 10^{-11} \\ 0 & 5 \times 10^{-11} \end{bmatrix}$$

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

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

and $\mathbf{A} + \Delta\mathbf{A}$ 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 $\|\Delta\mathbf{A}\|$. The distance from a nonsingular matrix **A** to the nearest singular matrix **S** is $1/\|\mathbf{A}^{-1}\|$.



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

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

In the right diagram, $\|\Delta\mathbf{A}\| > 1/\|\mathbf{A}^{-1}\|$. 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 $\mathbf{AX} = \mathbf{B}$ for the theoretical solution $\mathbf{X}$. Because of rounding errors during the calculations, the calculated solution $\mathbf{Z}$ is in general not the solution to the original system $\mathbf{AX} = \mathbf{B}$, but rather the solution to the perturbed system $(\mathbf{A} + \Delta\mathbf{A})\mathbf{Z} = \mathbf{B}$. The perturbation $\Delta\mathbf{A}$ satisfies $\|\Delta\mathbf{A}\| \leqslant \epsilon\|\mathbf{A}\|$, where ϵ is usually a very small number. In many cases, $\Delta\mathbf{A}$ will amount to less than one in the 10th digit of each element of $\mathbf{A}$.

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

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

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

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

where n is the dimension of $\mathbf{A}$. For the HP-15C, which carries 10 accurate digits,

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

In many applications, this accuracy may be adequate. When additional accuracy is desired, the computed solution $\mathbf{Z}$ 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 $\mathbf{Z}$ to the original system $\mathbf{AX} = \mathbf{B}$. $\mathbf{Z}$ 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

$$\mathbf{A}=\begin{bmatrix} 3\times10^{-40} & 1 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & -1 \end{bmatrix}.$$

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 $\|\mathbf{E}-\mathbf{S}\|/\|\mathbf{E}\|=\frac{1}{3}\times10^{-40}$. This means that $K(\mathbf{S})\geqslant3\times10^{40}$, so it's not surprising that the calculated $\mathbf{E}^{-1}$

$$\mathbf{E}^{-1}\approx\begin{bmatrix} -6.67\times10^{-11} & 1 & 10^{-10} \\ 0.8569 & 8.569\times10^{9} & -4.284\times10^{9} \\ 0.07155 & -4.284\times10^{9} & 2.142\times10^{9} \end{bmatrix}$$

is far from the true value

$$\mathbf{E}^{-1} = \begin{bmatrix} -2\times10^{-40} & 3 & -1 \\ 3 & -4\times10^{40} & 2\times10^{40} \\ -1 & 2\times10^{40} & -10^{40} \end{bmatrix}.$$

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,

$$\mathbf{LER} = \begin{bmatrix} 3\times10^{-10} & 1 & 2 \\ 1 & 10^{-30} & 10^{-30} \\ 2 & 10^{-30} & -10^{-30} \end{bmatrix},$$

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

$$(\mathbf{LER})^{-1} = \begin{bmatrix} -2\times10^{-30} & 3 & -1 \\ 3 & -4\times10^{30} & 2\times10^{30} \\ -1 & 2\times10^{30} & -10^{30} \end{bmatrix}.$$

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

$$\mathbf{E}^{-1} = \mathbf{R}(\mathbf{LER})^{-1}\mathbf{L} = \begin{bmatrix} -2\times10^{-40} & 3 & -1 \\ 3 & -4\times10^{40} & 2\times10^{40} \\ -1 & 2\times10^{40} & -10^{40} \end{bmatrix},$$

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 $\mathbf{EX} = \mathbf{B}$, where **E** is poorly scaled. When solving for **X**, replace the system $\mathbf{EX} = \mathbf{B}$ by a system $(\mathbf{LER})\mathbf{Y} = \mathbf{LB}$ 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 $\mathbf{X} = \mathbf{RY}$.

Preconditioning

Preconditioning is another method by which you can replace a difficult system, $\mathbf{EX} = \mathbf{B}$, by an easier one, $\mathbf{AX} = \mathbf{D}$, 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 $\mathbf{E}$ by $\mathbf{r}^T\mathbf{E}$ and the kth row of $\mathbf{B}$ by $\mathbf{r}^T\mathbf{B}$. Provided that no roundoff has occurred during the evaluation of these new rows, the new system matrix $\mathbf{A}$ should be better conditioned (farther from singular) than $\mathbf{E}$ was, but the system will still have the same solution $\mathbf{X}$ as before.

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

As an illustration of the preconditioning process, consider the system $\mathbf{EX} = \mathbf{B}$, 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 $\mathbf{X}$ 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 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 $a = 10^{-4}$,

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

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

Now replace the first row of $\mathbf{E}$ by

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

and the first row of $\mathbf{B}$ by $10^7\mathbf{r}^T\mathbf{B} = 10^7$. This gives a new system equation $\mathbf{AX} = \mathbf{D}$, 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 **E** and **A** has roughly the same norm as every other. Using this new system, the HP-15C calculates the solution

$$\mathbf{X} = \begin{bmatrix} 2000.000080 \\ 1999.999980 \\ 1999.999980 \\ 1999.999980 \\ 1999.999980 \end{bmatrix}, \text{ with } \mathbf{AX} = \begin{bmatrix} 10^7 \\ -10^{-5} \\ -9\times10^{-6} \\ 0 \\ 0 \end{bmatrix}.$$

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

Sometimes the elements of a nearly singular matrix **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 **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 **X** of observed data and a vector **y** of n observations from which you must find a vector **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 **b** to the *normal equations*

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

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

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

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

Then

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

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

$$\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{y}.$$

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

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

subject to the constraints

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

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

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

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

$$\mathbf{Q}\mathbf{r} = \mathbf{Q}\mathbf{y} - \mathbf{Q}\mathbf{X}\mathbf{b} = \mathbf{Q}\mathbf{y} - \mathbf{U}\mathbf{b}.$$

The upper-triangular matrix **U** and the product **Qy** can be written as

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

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 **V**, which you can then partition as

$$\mathbf{V} = \begin{bmatrix} \hat{\mathbf{U}} & \mathbf{g} \\ \mathbf{0} & q \\ \mathbf{0} & 0 \end{bmatrix} \begin{matrix} (p \text{ rows}) \\ (1 \text{ row}) \\ (n-p-1 \text{ rows}) \end{matrix}$$

(1 column)

(p columns)

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

2. Solve the following system for **b**:

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

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

For the weighted least-squares problem, replace **X** and **y** by **WX** and **Wy**, where **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 $\|\mathbf{C}\mathbf{b} - \mathbf{d}\| < t$. 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 $w\|\mathbf{C}\mathbf{b} - \mathbf{d}\| > \|\mathbf{y}\|$ unless $\|\mathbf{C}\mathbf{b} - \mathbf{d}\| < t$.

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

- No **b** exists for which $\|\mathbf{C}\mathbf{b} - \mathbf{d}\| < t$.
- The leading columns of **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 **b**

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

using any small nonzero number q. If the computed solution **b** satisfies $\mathbf{C}\mathbf{b} \approx \mathbf{d}$, 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

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

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

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

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

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

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)]	010u 44 24	Skips next step at last element.
[f] [USER]		
[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 $\mathbf{X}$, then performs one stage iterative refinement to improve the solution. The program uses four matrices:

Matrix	A	B	C	D
Input	System Matrix	Right-Hand Matrix		
Output	System Matrix	Corrected Solution	Uncorrected Solution	*LU* Form of **A**

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	
f LBL A	001-42,21,11	
RCL MATRIX A	002-45,16,11	
STO MATRIX D	003-44,16,14	Stores system matrix in **D**.
RCL MATRIX B	004-45,16,12	
RCL MATRIX D	005-45,16,14	
f RESULT C	006-42,26,13	
÷	007- 10	Calculates uncorrected solution, **C**.
f RESULT B	008-42,26,12	
f MATRIX 6	009-42,16, 6	Calculates residual, **B**.
RCL MATRIX D	010-45,16,14	
÷	011- 10	Calculates correction, **B**.
RCL MATRIX C	012-45,16,13	
+	013- 40	Calculates refined solution, **B**.
g RTN	014- 43 32	
g P/R		Run mode.

Label used: A.

Matrices used: **A**, **B**, **C**, and **D**.

To use this program:

1. Dimension matrix **A** according to the system matrix and store those elements in **A**.
2. Dimension matrix **B** according to the right-hand matrix and store those elements in **B**.
3. Press GSB A to calculate the corrected solution in matrix **B**.

Example: Use the residual correction program to calculate the inverse of matrix **A** for

$$\mathbf{A} = \begin{bmatrix} 33 & 16 & 72 \\ -24 & -10 & -57 \\ -8 & -4 & -17 \end{bmatrix}.$$

The theoretical inverse of **A** is

$$\mathbf{A}^{-1} = \begin{bmatrix} -29/3 & -8/3 & -32 \\ 8 & 5/2 & 51/2 \\ 8/3 & 2/3 & 9 \end{bmatrix}.$$

Find the inverse by solving $\mathbf{AX} = \mathbf{B}$ for **X**, where **B** is a 3×3 identity matrix.

First, enter the program from above. Then, in Run mode, enter the elements into matrix **A** (the system matrix) and matrix **B** (the right-hand, identity matrix). Press [GSB] [A] to execute the program.

Recall the elements of the uncorrected solution, matrix **C**:

$$\mathbf{C} = \begin{bmatrix} -9.666666881 & -2.666666726 & -32.00000071 \\ 8.000000167 & 2.500000046 & 25.50000055 \\ 2.666666728 & 0.6666666836 & 9.000000203 \end{bmatrix}.$$

This solution is correct to seven digits. The accuracy is well within that predicted by the equation on page 103.

$$(\text{number of correct digits}) \geqslant 9 - \log(\|\mathbf{A}\|\,\|\mathbf{C}\|) - \log(3) \approx 4.8\,.$$

Recall the elements of the corrected solution, matrix **B**:

$$\mathbf{B} = \begin{bmatrix} -9.666666667 & -2.666666667 & -32.00000000 \\ 8.000000000 & 2.500000000 & 25.50000000 \\ 2.666666667 & 0.6666666667 & 9.000000000 \end{bmatrix}.$$

One iteration of refinement yields 10 correct digits in this case.

Solving a System of Nonlinear Equations

Consider a system of p nonlinear equations in p unknowns:

$$f_i(x_1, x_2, ..., x_p) = 0 \qquad \text{for } i = 1, 2, ..., p$$

for which the solution $x_1, x_2, ..., x_p$ is sought.

Let

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_p(\mathbf{x}) \end{bmatrix}, \text{ and } \mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_{11}(\mathbf{x}) \ldots F_{1p}(\mathbf{x}) \\ F_{21}(\mathbf{x}) \ldots F_{2p}(\mathbf{x}) \\ \vdots \qquad \vdots \\ F_{p1}(\mathbf{x}) \ldots F_{pp}(\mathbf{x}) \end{bmatrix},$$

where

$$F_{ij}(\mathbf{x}) = \frac{\partial}{\partial x_j} f_i(\mathbf{x}) \qquad \text{for } i, j = 1, 2, ..., p.$$

The system of equations can be expressed as $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Newton's method starts with an initial guess $\mathbf{x}^{(0)}$ to a root $\mathbf{x}$ of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ and calculates

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{F}(\mathbf{x}^{(k)}))^{-1}\mathbf{f}(\mathbf{x}^{(k)}) \qquad \text{for } k = 0, 1, 2, ...$$

until $\mathbf{x}^{(k+1)}$ converges.

The program in the following example performs one iteration of Newton's method. The computations are performed as

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{d}^{(k)},$$

where $\mathbf{d}^{(k)}$ is the solution to the $p \times p$ linear system

$$\mathbf{F}(\mathbf{x}^{(k)})\mathbf{d}^{(k)} = \mathbf{f}(\mathbf{x}^{(k)}).$$

The program displays the Euclidean lengths of $\mathbf{f}(\mathbf{x}^{(k)})$ and the correction $\mathbf{d}^{(k)}$ at the end of each iteration.

Example: For the random variable y having a normal distribution with unknown mean m and variance v^2, construct an unbiased test of the hypothesis that $v^2 = v_0^2$ versus the alternative that $v^2 \neq v_0^2$ for a particular value v_0^2.

For a random sample of y consisting of $y_1, y_2, ..., y_n$, an unbiased test rejects the hypothesis if

$$s_n < x_1 v_0^2 \qquad \text{or} \qquad s_n > x_2 v_0^2,$$

where

$$s_n = \sum_{i=1}^{n} (y_i - \bar{y})^2 \quad \text{and} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \quad ,$$

for some constants x_1 and x_2.

If the size of the test is a $(0 < a < 1)$, you can find x_1 and x_2 by solving the system of equations $f_1(\mathbf{x}) = f_2(\mathbf{x}) = 0$, where

$$f_1(\mathbf{x}) = (n-1)\ln(x_2/x_1) + x_1 - x_2$$

$$f_2(\mathbf{x}) = \int_{x_1}^{x_2} (w/2)^m \exp(-w/2)dw \quad - \quad 2(1-a)\Gamma(m+1).$$

Here $x_2 > x_1 > 0$, a and n are known $(n > 1)$, and $m = (n-1)/2 - 1$.

An initial guess for (x_1, x_2) is

$$x_1^{(0)} = \chi^2_{n-1,\,a/2} \quad \text{and} \quad x_2^{(0)} = \chi^2_{n-1,\,1-a/2}$$

where $\chi^2_{d,p}$ is the pth percentile of the chi-square distribution with d degrees of freedom.

For this example,

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} 1-(n-1)/x_1 & (n-1)/x_2 - 1 \\ -(x_1/2)^m \exp(-x_1/2) & (x_2/2)^m \exp(-x_2/2) \end{bmatrix} .$$

Enter the following program:

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	
f LBL A	001-42,21,11	
2	002- 2	
ENTER	003- 36	
f DIM C	004-42,23,13	Dimensions **F** matrix to 2×2.
1	005- 1	
f DIM B	006-42,23,12	Dimensions **f** matrix to 2×1.

Keystrokes	Display	
GSB B	007- 32 12	Calculates **f** and **F**.
RCL MATRIX A	008-45,16,11	
RCL MATRIX B	009-45,16,12	
RCL MATRIX C	010-45,16,13	
f RESULT D	011-42,26,14	
÷	012- 10	Calculates $\mathbf{d}^{(k)}$.
f RESULT A	013-42,26,11	
−	014- 30	Calculates $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{d}^{(k)}$.
g LSTx	015- 43 36	
f MATRIX 8	016-42,16, 8	Calculates $\|\mathbf{d}^{(k)}\|_F$.
RCL MATRIX B	017-45,16,12	
f MATRIX 8	018-42,16, 8	Calculates $\|\mathbf{f}(\mathbf{x}^{(k)})\|_F$.
g RTN	019- 43 32	
f LBL B	020-42,21,12	Routine to calculate **f** and **F**.
f MATRIX 1	021-42,16, 1	
f USER RCL A f USER	022u 45 11	
STO 4	023- 44 4	Stores $x_1^{(k)}$ in R_4.
f USER RCL A f USER	024u 45 11	Skips next line for last element.
STO 5	025- 44 5	Stores $x_2^{(k)}$ in R_5.
STO 5	026- 44 5	
−	027- 30	Calculates $x_1 - x_2$.
RCL 5	028- 45 5	
RCL ÷ 4	029-45,10, 4	
g LN	030- 43 12	Calculates $\ln(x_2/x_1)$.
RCL 2	031- 45 2	
1	032- 1	
−	033- 30	
×	034- 20	Calculates $(n-1)\ln(x_2/x_1)$.
+	035- 40	Calculates f_1.
STO B	036- 44 12	Stores f_1 in **B**.
1	037- 1	

Keystrokes	Display	
RCL 2	038- 45 2	
1	039- 1	
−	040- 30	
RCL ÷ 4	041-45,10, 4	Calculates $(n-1)/x_1$.
−	042- 30	Calculates F_{11}.
f USER STO C	043u 44 13	Stores F_{11} in **C**.
f USER		
RCL 2	044- 45 2	
1	045- 1	
−	046- 30	
RCL ÷ 5	047-45,10, 5	Calculates $(n-1)/x_2$.
1	048- 1	
−	049- 30	Calculates F_{12}.
f USER STO C	050u 44 13	Stores F_{12} in **C**.
f USER		
RCL 4	051- 45 4	
RCL 5	052- 45 5	
f ∫ C	053-42,20,13	Calculates integral.
RCL 3	054- 45 3	
1	055- 1	
−	056- 30	
2	057- 2	
×	058- 20	Calculates $2(a-1)$.
RCL 2	059- 45 2	
3	060- 3	
−	061- 30	
2	062- 2	
÷	063- 10	Calculates m.
f x!	064- 42 0	Calculates $\Gamma(m+1)$.
×	065- 20	
+	066- 40	Calculates f_2.
STO B	067- 44 12	Stores f_2 in **B**.
RCL 4	068- 45 4	
GSB C	069- 32 13	
CHS	070- 16	Calculates F_{21}.
f USER STO C	071u 44 13	Stores F_{21} in **C**.
f USER		

Keystrokes	Display	
RCL 5	072- 45 5	
GSB C	073- 32 13	Calculates F_{22}.
f USER STO C	074u 44 13	Stores F_{22} in C.
f USER		
g RTN	075- 43 32	Skips this line.
g RTN	076- 43 32	
f LBL C	077-42,21,13	Integrand routine.
2	078- 2	
÷	079- 10	
CHS	080- 16	
e^x	081- 12	Calculates $e^{-x/2}$.
g LSTx	082- 43 36	
CHS	083- 16	
RCL 2	084- 45 2	
3	085- 3	
-	086- 30	
2	087- 2	
÷	088- 10	Calculates m.
y^x	089- 14	
×	090- 20	Calculates $(x/2)^m e^{-x/2}$.
g RTN	091- 43 32	

Labels used: A, B, and C.

Registers used: R_0 (row), R_1 (column), R_2 (n), R_3 (a), R_4 ($x_1^{(k)}$), and R_5 ($x_2^{(k)}$).

Matrices used: **A** ($\mathbf{x}^{(k+1)}$), **B** ($\mathbf{f}(\mathbf{x}^{(k)})$), **C** ($\mathbf{F}(\mathbf{x}^{(k)})$), and **D** ($\mathbf{d}^{(k)}$).

Now run the program. For example, choose the values $n = 11$ and $a = 0.05$. The suggested initial guesses are $x_1^{(0)} = 3.25$ and $x_2^{(0)} = 20.5$. Remember that the display format affects the uncertainty of the integral calculation.

Keystrokes	Display	
g P/R		Run mode.
5 f DIM (i)	5.0000	Reserves R_0 through R_5.
11 STO 2	11.0000	Stores n in R_2.

Keystrokes	Display		
.05 [STO] 3	0.0500		Stores a in R_3.
2 [ENTER] 1	1		
[f] [DIM] [A]	1.0000		Dimensions **A** to 2×1.
[f] [USER]	1.0000		Activates User mode.
[f] [MATRIX] 1	1.0000		
3.25 [STO] [A]	3.2500		Stores $x_1^{(0)}$ from chi-square distribution.
20.5 [STO] [A]	20.5000		Stores $x_2^{(0)}$ from chi-square distribution.
[f] [SCI] 4	2.0500	01	Sets display format.
[A]	1.1677	00	Displays norm of $\mathbf{f}(\mathbf{x}^{(0)})$.
[R↓]	1.0980	00	Displays norm of correction $\mathbf{d}^{(0)}$.
[RCL] [A]	3.5519	00	Recalls $x_1^{(1)}$.
[RCL] [A]	2.1556	01	Recalls $x_2^{(1)}$

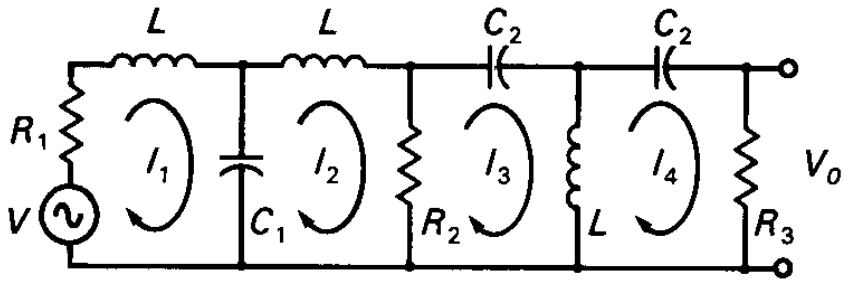
By repeating the last four steps, you will obtain these results:

k	$\|\mathbf{f}(\mathbf{x}^{(k)})\|_F$	$\|\mathbf{d}^{(k)}\|_F$	$x_1^{(k+1)}$	$x_2^{(k+1)}$
			3.2500	20.500
0	1.168	1.098	3.5519	21.556
1	1.105×10^{-1}	1.740×10^{-1}	3.5169	21.726
2	1.918×10^{-3}	2.853×10^{-3}	3.5162	21.729
3	6.021×10^{-7}	9.542×10^{-7}	3.5162	21.729

This accuracy is sufficient for constructing the statistical test. (Press [f] [FIX] 4 to reset the display format and [f] [USER] to deactivate User mode.)

Solving a Large System of Complex Equations

Example: Find the output voltage at a radian frequency of $\omega = 15 \times 10^3$ rad/s for the filter network shown below.



$V = 10$ volts
$R_1 = 100$ ohms
$R_2 = 10^6$ ohms
$R_3 = 10^5$ ohms
$L = 10^{-2}$ henry
$C_1 = 25 \times 10^{-8}$ farad
$C_2 = 25 \times 10^{-6}$ farad

Describe the circuit using loop currents:

$$\begin{bmatrix} (R_1+i\omega L-i/\omega C_1) & (i/\omega C_1) & 0 & 0 \\ (i/\omega C_1) & (R_2+i\omega L-i/\omega C_1) & (-R_2) & 0 \\ 0 & (-R_2) & (R_2-i/\omega C_2+i\omega L) & (-i\omega L) \\ 0 & 0 & (-i\omega L) & (R_3+i\omega L-i/\omega C_2) \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Solve this complex system for I_1, I_2, I_3, and I_4. Then $V_O = (R_3)(I_4)$.

Because this system is too large to solve using the standard method for a system of complex equations, this alternate method (described in the owner's handbook) is used. First, enter the system matrix into matrix **A** in complex form and calculate its inverse. Note that $\omega L = 150$, $1/\omega C_1 = 800/3$, and $1/\omega C_2 = 8/3$.

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	Clears program memory.

Keystrokes	Display			
g P/R				Run mode.
0 f DIM (i)	0.0000			Provides maximum matrix memory.
f MATRIX 0	0.0000			Dimensions all matrices to 0×0.
4 ENTER 8	8			
f DIM A	8.0000			Dimensions matrix **A** to 4×8.
f MATRIX 1	8.0000			
f USER	8.0000			Activates User mode.
100 STO A	100.0000			Stores $\mathrm{Re}(a_{11})$.
150 ENTER	150.0000			
800 ENTER 3 ÷	266.6667			
- STO A	-116.6667			Stores $\mathrm{Im}(a_{11})$.
⋮				
150 ENTER	150.0000			
8 ENTER 3 ÷	2.6667			
- STO A	147.3333			Stores $\mathrm{Im}(a_{44})$.
RCL MATRIX A	A	4	8	
f Py,x	A	8	4	Transforms $\mathbf{A}^C$ to $\mathbf{A}^P$.
f MATRIX 2	A	8	8	Transforms $\mathbf{A}^P$ to $\tilde{\mathbf{A}}$.
STO RESULT	A	8	8	
f 1/x	A	8	8	Calculates inverse of $\tilde{\mathbf{A}}$ in **A**.

Delete the second half of the rows of **A** to provide space to store the right-hand matrix **B**.

Keystrokes	Display	
4 ENTER 8	8	
f DIM A	8.0000	Redimensions matrix **A** to 4×8.
4 ENTER 2	2	
f DIM B	2.0000	Dimensions matrix **B** to 4×2.

Keystrokes	Display			
f MATRIX 1	2.0000			
10 STO B	10.0000			Stores Re(V). (Other elements are 0.)
RCL MATRIX A	A	4	8	
RCL MATRIX B	b	4	2	
f Py,x	b	8	1	Transforms $\mathbf{B}^C$ to $\mathbf{B}^P$.
f MATRIX 2	b	8	2	Transforms $\mathbf{B}^P$ to $\tilde{\mathbf{B}}$.
f RESULT C	b	8	2	
×	C	4	2	Calculates solution in **C**.
f MATRIX 4	C	2	4	Calculates transpose.
f MATRIX 2	C	2	8	Transforms **C** to $\tilde{\mathbf{C}}$.
1 ENTER 8	8			
f DIM C	8.0000			Redimensions matrix **C** to 1×8.
RCL RESULT	C	1	8	
f MATRIX 4	C	8	1	Calculates transpose.
g Cy,x	C	4	2	Transforms $\mathbf{C}^P$ to $\mathbf{C}^C$.

Matrix C contains the desired values of I_1, I_2, I_3, and I_4 in rectangular form. Their phasor forms are easy to compute:

Keystrokes	Display			
f MATRIX 1	C	4	2	Resets R_0 and R_1.
f SCI 4	C	4	2	
RCL C	1.9950		-04	Recalls Re(I_1).
RCL C	4.0964		-03	Recalls Im(I_1).
x≷y g →P	4.1013		-03	Displays $\lvert I_1 \rvert$.
x≷y	8.7212		01	Displays Arg(I_1) in degrees.
RCL C	-1.4489		-03	
RCL C	-3.5633		-02	
x≷y g →P	3.5662		-02	Displays $\lvert I_2 \rvert$.
x≷y	-9.2328		01	
RCL C	-1.4541		-03	
RCL C	-3.5633		-02	
x≷y g →P	3.5662		-02	Displays $\lvert I_3 \rvert$.

Keystrokes	Display		
[x≷y]	-9.2337	01	
[RCL] [C]	5.3446	-05	
[RCL] [C]	-2.2599	-06	
[x≷y] [g] [→P]	5.3494	-05	Displays $\|I_4\|$.
[x≷y]	-2.4212	00	
[x≷y] [EEX] 5 [×]	5.3494	00	Calculates $\|V_O\| = (R_3)\|I_4\|$.
[f] [FIX] 4	5.3494		
[f] [USER]	5.3494		Deactivates User mode.

The output voltage is $5.3494 \angle -2.4212°$.

Least-Squares Using Normal Equations

The unconstrained least-squares problem is known in statistical literature as *multiple linear regression*. It uses the linear model

$$y = \sum_{j=1}^{p} b_j x_j + r.$$

Here, b_1, ..., b_p are the unknown parameters, x_1, ..., x_p are the independent (or explanatory) variables, y is the dependent (or response) variable, and r is the random error having expected value $E(r) = 0$, variance σ^2.

After making n observations of y and $x_1, x_2, ..., x_p$, this problem can be expressed as

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{r}$$

where $\mathbf{y}$ is an n-vector, $\mathbf{X}$ is an $n \times p$ matrix, and $\mathbf{r}$ is an n-vector consisting of the unknown random errors satisfying $E(\mathbf{r}) = \mathbf{0}$ and $\text{Cov}(\mathbf{r}) = E(\mathbf{r}\mathbf{r}^T) = \sigma^2\mathbf{I}_n$.

If the model is correct and $\mathbf{X}^T\mathbf{X}$ has an inverse, then the calculated least-squares solution $\hat{\mathbf{b}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ has the following properties:

- $E(\hat{\mathbf{b}}) = \mathbf{b}$, so that $\hat{\mathbf{b}}$ is an unbiased estimator of $\mathbf{b}$.
- $\text{Cov}(\hat{\mathbf{b}}) = E((\hat{\mathbf{b}} - \mathbf{b})^T(\hat{\mathbf{b}} - \mathbf{b})) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$, the covariance matrix of the estimator $\hat{\mathbf{b}}$.

- $E(\hat{\mathbf{r}}) = 0$, where $\hat{\mathbf{r}} = \mathbf{y} - \mathbf{X}\hat{\mathbf{b}}$ is the vector of residuals.
- $E(\|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}\|_F^2) = (n - p)\sigma^2$, so that $\hat{\sigma}^2 = \|\hat{\mathbf{r}}\|_F^2/(n - p)$ is an unbiased estimator for σ^2. You can estimate $\text{Cov}(\hat{\mathbf{b}})$ by replacing σ^2 by $\hat{\sigma}^2$.

The total sum of squares $\|\mathbf{y}\|_F^2$ can be partitioned according to

$$\begin{aligned}\|\mathbf{y}\|_F^2 &= \mathbf{y}^T\mathbf{y} \\ &= (\mathbf{y} - \mathbf{X}\hat{\mathbf{b}} + \mathbf{X}\hat{\mathbf{b}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}} + \mathbf{X}\hat{\mathbf{b}}) \\ &= (\mathbf{y} - \mathbf{X}\hat{\mathbf{b}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}) - 2\hat{\mathbf{b}}^T\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}) + (\mathbf{X}\hat{\mathbf{b}})^T(\mathbf{X}\hat{\mathbf{b}}) \\ &= \|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}\|_F^2 + \|\mathbf{X}\hat{\mathbf{b}}\|_F^2 \\ &= \begin{pmatrix}\text{Residual} \\ \text{Sum of Squares}\end{pmatrix} + \begin{pmatrix}\text{Regression} \\ \text{Sum of Squares}\end{pmatrix}.\end{aligned}$$

When the model is correct,

$$E(\|\mathbf{X}\hat{\mathbf{b}}\|_F^2/p) = \sigma^2 + \|\mathbf{X}\mathbf{b}\|_F^2/p > \sigma^2$$

and

$$E(\|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}\|_F^2/(n - p)) = \sigma^2$$

for $\mathbf{b} \neq \mathbf{0}$. When the simpler model $y = r$ is correct, both of these expectations equal σ^2.

You can test the hypothesis that the simpler model is correct (against the alternative that the original model is correct) by calculating the F ratio

$$F = \frac{\|\mathbf{X}\hat{\mathbf{b}}\|_F^2/p}{\|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}\|_F^2/(n - p)}.$$

F will tend to be larger when the original model is true ($\mathbf{b} \neq \mathbf{0}$) than when the simpler model is true ($\mathbf{b} = \mathbf{0}$). You reject the hypothesis when F is sufficiently large.

If the random errors have a normal distribution, the F ratio has a central F distribution with p and $(n - p)$ degrees of freedom if $\mathbf{b} = \mathbf{0}$, and a noncentral distribution if $\mathbf{b} \neq \mathbf{0}$. A statistical test of the hypothesis (with probability α of incorrectly rejecting the hypothesis) is to reject the hypothesis if the F ratio is larger than the 100α percentile of the central F distribution with p and $(n - p)$

degrees of freedom; otherwise, accept the hypothesis.

The following program fits the linear model to a set of n data points $x_{i1}, x_{i2}, ..., x_{ip}, y_i$ by the method of least-squares. The parameters b_1, $b_2, ..., b_p$ are estimated by the solution $\hat{\mathbf{b}}$ to the normal equations $\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$. The program also estimates σ^2 and the parameter covariance matrix $\text{Cov}(\hat{\mathbf{b}})$. The regression and residual sums of squares (*Reg SS* and *Res SS*) and the residuals are also calculated.

The program requires two matrices:

Matrix **A**: $n \times p$ with row i $(x_{i1}, x_{i2}, ..., x_{ip})$
for $i = 1, 2, ..., n$.
Matrix **B**: $n \times 1$ with element i (y_i) for $i = 1, 2, ..., n$.

The program output is:

Matrix **A**: unchanged.
Matrix **B**: $n \times 1$ containing the residuals from the fit $(y_i - \hat{b}_1 x_{i1} - ... - \hat{b}_p x_{ip})$ for $i = 1, 2, ..., n$, where $\hat{b}_i$ is the estimate for b_i.
Matrix **C**: $p \times p$ covariance matrix of the parameter estimates.
Matrix **D**: $p \times 1$ containing the parameter estimates $\hat{b}_1, ..., \hat{b}_p$.
T-register: contains an estimate of σ^2.
Y-register: contains the regression sum of squares (*Reg SS*).
X-register: contains the residual sum of squares (*Res SS*).

The analysis of variance (ANOVA) table below partitions the total sum of squares (*Tot SS*) into the regression and the residual sums of squares. You can use the table to calculate the F ratio.

ANOVA Table

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Ratio
Regression	p	*Reg SS*	$\frac{(Reg\ SS)}{p}$	$\frac{(Reg\ MS)}{(Res\ MS)}$
Residual	$n-p$	*Res SS*	$\frac{(Res\ SS)}{(n-p)}$	
Total	n	*Tot SS*		

The program calculates the regression sum of squares *unadjusted* for the mean because a constant term may not be in the model. To include a constant term, include in the model a variable that is identically equal to one. The corresponding parameter is then the constant term.

To calculate the *mean-adjusted* regression sum of squares for a model containing a constant term, first use the program to fit the model and to find the unadjusted regression sum of squares. Then fit the simpler model $y = b_1 + r$ by dropping all variables but the one identically equal to one (b_1, for example) and find the regression sum of squares for this model, $(Reg\ SS)_C$. The mean-adjusted regression sum of squares $(Reg\ SS)_A = Reg\ SS - (Reg\ SS)_C$. Then the ANOVA table becomes:

ANOVA Table

Source	Degrees of Freedom	Sum of Squares	Mean Square	*F* Ratio
Regression | Constant	$p-1$	$(Reg\ SS)_A$	$\dfrac{(Reg\ SS)_A}{(p-1)}$	$\dfrac{(Reg\ MS)_A}{(Res\ MS)}$
Constant	1	$(Reg\ SS)_C$	$(Res\ SS)_C$	
Residual	$n-p$	$Res\ SS$	$\dfrac{(Res\ SS)}{(n-p)}$	
Total	n	$Tot\ SS$		

You can then use the F ratio to test whether the full model fits data significantly better than the simpler model $y = b_1 + r$.

You may want to perform a series of regressions, dropping independent variables between each. To do this, order the variables in the reverse order that they will be dropped from the model. They can be dropped by transposing the matrix **A**, redimensioning **A** to have fewer rows, and then transposing **A** once again.

You will need the original dependent variable data for each regression. If there is not enough room to store the original data in matrix **E**, you can compute it from the output of the regression fit. A subroutine has been included to do this.

This program has the following characteristics:

- If the entire program is keyed into program memory, the sizes of n and p are required to satisfy $n \geqslant p$ and $(n+p)(p+1) \leqslant 56$. That is,

if p is	1	2	3	4
then n_{max} is	27	16	11	7

  This assumes that only data storage registers R_0 and R_1 are allocated. If subroutine "B" is omitted, then $n \geqslant p$ and $(n+p)(p+1) \leqslant 58$. That is,

if p is	1	2	3	4
then n_{max} is	28	17	11	7

- Even though subroutine "B" uses the residual function with its extended precision, the computed dependent variable data may not exactly agree with the original data. The agreement will usually be close enough for statistical estimation and tests. If more accuracy is desired, the original data can be reentered into matrix **B**.

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	
f LBL A	001-42,21,11	Program to fit model.
RCL MATRIX B	002-45,16,12	
f MATRIX 8	003-42,16, 8	
g x^2	004- 43 11	Calculates *Tot SS*.
RCL MATRIX A	005-45,16,11	
ENTER	006- 36	
f RESULT C	007-42,26,13	
f MATRIX 5	008-42,16, 5	Calculates $\mathbf{C} = \mathbf{A}^T\mathbf{A}$.
g LSTx	009- 43 36	
RCL MATRIX B	010-45,16,12	
f RESULT D	011-42,26,14	
f MATRIX 5	012-42,16, 5	Calculates $\mathbf{D} = \mathbf{A}^T\mathbf{B}$.
x≷y	013- 34	

Keystrokes	Display	
[÷]	014- 10	Calculates parameters in **D**.
[RCL] [MATRIX] [A]	015-45,16,11	
[x≷y]	016- 34	
[f] [RESULT] [B]	017-42,26,12	
[f] [MATRIX] 6	018-42,16, 6	Calculates residuals of fit in **B**.
[f] [MATRIX] 8	019-42,16, 8	
[g] [x^2]	020- 43 11	Calculates *Res SS*.
[RCL] [DIM] [A]	021-45,23,11	
[−]	022- 30	
[÷]	023- 10	Calculates σ^2 estimate.
[ENTER]	024- 36	
[ENTER]	025- 36	
[RCL] [MATRIX] [C]	026-45,16,13	
[f] [RESULT] [C]	027-42,26,13	
[÷]	028- 10	Calculates covariance matrix in **C**.
[g] [R↓]	029- 43 33	
[RCL] [MATRIX] [B]	030-45,16,12	
[f] [MATRIX] 8	031-42,16, 8	
[g] [x^2]	032- 43 11	
[−]	033- 30	Calculates *Reg SS*.
[g] [LSTx]	034- 43 36	Returns *Res SS*.
[g] [RTN]	035- 43 32	
[f] [LBL] [B]	036-42,21,12	Subroutine to reconstruct dependent variable data.
[RCL] [MATRIX] [A]	037-45,16,11	
[RCL] [MATRIX] [D]	038-45,16,14	
[CHS]	039- 16	
[f] [RESULT] [B]	040-42,26,12	
[f] [MATRIX] 6	041-42,16, 6	Calculates $\mathbf{B} = \mathbf{B} + \mathbf{AD}$.
[RCL] [MATRIX] [D]	042-45,16,14	
[CHS]	043- 16	
[g] [RTN]	044- 43 32	

Labels used: A and B.

Registers used: R_0 and R_1.

Matrices used: **A**, **B**, **C**, and **D**.

To use this program:

1. Press 1 [f] [DIM] [(i)] to reserve registers R_0 and R_1.
2. Dimension matrix **A** according to the number of observations n and the number of parameters p by pressing n [ENTER] p [f] [DIM] [A].
3. Dimension matrix **B** according to the number of observations n (and one column) by pressing n [ENTER] 1 [f] [DIM] [B].
4. Press [f] [MATRIX] 1 to set registers R_0 and R_1.
5. Press [f] [USER] to activate User mode.
6. For each observation, store the values of the p variables in a row of matrix **A**. Repeat this for the n observations.
7. Store the values of the dependent variable in matrix **B**.
8. Press [A] to calculate and display the *Res SS*. The Y-register contains the *Reg SS* and the T-register contains the σ^2 estimate.
9. Press [RCL] [D] to observe each of the p parameter estimates.
10. If desired, press [B] to recalculate the dependent variable data in matrix **B**.

Example: Compare two regression models of the annual change in the consumer price index (CPI) using the annual change in the producer price index (PPI) and the unemployment rate (UR):

$$y = b_1 + b_2x_2 + b_3x_3 + r \qquad \text{and} \qquad y = b_1 + b_2x_2 + r,$$

where y, x_2, and x_3 represent CPI, PPI, and UR (all as percentages). Use the following data from the U.S.:

Year	CPI	PPI	UR
1969	5.4	3.9	3.5
1970	5.9	3.7	4.9
1971	4.3	3.3	5.9
1972	3.3	4.5	5.6
1973	6.2	13.1	4.9
1974	11.0	18.9	5.6
1975	9.1	9.2	8.5
1976	5.8	4.6	7.7
1977	6.5	6.1	7.0
1978	7.6	7.8	6.0
1979	11.5	19.3	5.8

Keystrokes	Display	
g P/R		Run mode.
f MATRIX 0		
11 ENTER 3	**3**	
f DIM A	**3.0000**	Dimensions **A** as 11×3.
11 ENTER 1	**1**	
f DIM B	**1.0000**	Dimensions **B** as 11×1.
f MATRIX 1	**1.0000**	
f USER	**1.0000**	
1 STO A	**1.0000**	Enters independent variable data.
3.9 STO A	**3.9000**	
3.5 STO A	**3.5000**	
⋮	⋮	
1 STO A	**1.0000**	
19.3 STO A	**19.3000**	
5.8 STO A	**5.8000**	
5.4 STO B	**5.4000**	Enters dependent variable data.
5.9 STO B	**5.9000**	
⋮	⋮	
11.5 STO B	**11.5000**	
A f FIX 9	**13.51217504**	*Res SS* for full model.
R↓	**587.9878252**	*Reg SS* for full model.

Keystrokes	Display	
R↓ R↓	1.689021880	σ^2 estimate.
RCL D	1.245864326	b_1 estimate.
RCL D	0.379758235	b_2 estimate.
RCL D	0.413552218	b_3 estimate.
B	d 3 1	Recalculates dependent data.
RCL MATRIX A	A 11 3	
f MATRIX 4	A 3 11	
2 ENTER 11	11	
f DIM A	11.00000000	Drops last column of **A**.
RCL MATRIX A	A 2 11	
f MATRIX 4	A 11 2	New **A** matrix.
A	16.78680552	*Res SS* for reduced model.
R↓	584.7131947	*Reg SS* for reduced model.
R↓ R↓	1.865200613	σ^2 estimate.
RCL D	3.701730745	b_1 estimate.
RCL D	0.380094935	b_2 estimate.
B	d 2 1	Recalculates dependent data.
RCL MATRIX A	A 11 2	
f MATRIX 4	A 2 11	
1 ENTER 11	11	
f DIM A	11.00000000	Drops next column of **A**.
RCL MATRIX A	A 1 11	
f MATRIX 4	A 11 1	New **A** matrix.
A	68.08545454	*Res SS*.
R↓	533.4145457	*Reg SS* for constant.
R↓ R↓	6.808545454	σ^2 estimate.
RCL D	6.963636364	b_1 estimate.
f USER	6.963636364	Deactivates User mode.
f FIX 4	6.9636	

The *Reg SS* for the PPI variable adjusted for the constant term is (*Reg SS* for reduced model) − (*Reg SS* for constant) = 51.29864900.

The *Reg SS* for the UR variable adjusted for the PPI variable and the constant term is

(*Reg SS* for full model) − (*Reg SS* for reduced model) = 3.274630500.

Now construct the following ANOVA table:

Source	Degrees of Freedom	Sum of Squares	Mean Square	*F* Ratio
UR| PPI, Constant	1	3.2746305	3.2746305	1.939
PPI| Constant	1	51.2986490	51.2986490	30.37
Constant	1	533.4145457	533.4145457	315.8
Residual (full model)	8	13.5121750	1.68902188	
Total	11	601.5000002		

The *F* ratio for the unemployment rate, adjusted for the producer price index change and the constant is not statistically significant at the 10-percent significance level ($\alpha = 0.1$). Including the unemployment rate in the model does not significantly improve the CPI fit.

However, the *F* ratio for the producer price index adjusted for the constant *is* significant at the 0.1-percent level ($\alpha = 0.001$). Including the PPI in the model does improve the CPI fit.

Least-Squares Using Successive Rows

This program uses orthogonal factorization to solve the least-squares problem. That is, it finds the parameters $b_1, \ldots, b_p$ that minimize the sum of squares $\|\mathbf{r}\|_F^2 = (\mathbf{y} - \mathbf{Xb})^T(\mathbf{y} - \mathbf{Xb})$ given the model data

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

The program does this for successively increasing values of n, although the solution $\mathbf{b} = \mathbf{b}^{(n)}$ is meaningful only when $n \geqslant p$.

It is possible to factor the augmented $n \times (p+1)$ matrix $[\mathbf{X} \quad \mathbf{y}]$ into $\mathbf{Q}^T\mathbf{V}$, where $\mathbf{Q}$ is an orthogonal matrix,

$$\mathbf{V} = \begin{bmatrix} \hat{\mathbf{U}} & \mathbf{g} \\ \mathbf{0} & q \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{matrix} (p \text{ rows}) \\ (1 \text{ row}), \\ (n-p-1 \text{ rows}) \end{matrix}$$

and $\hat{\mathbf{U}}$ is an upper-triangular matrix. If this factorization results from including n rows $\mathbf{r}_m = (x_{m1}, x_{m2}, \dots, x_{mp}, y_m)$ for $m = 1, 2, \dots, n$ in $[\mathbf{X} \quad \mathbf{y}]$, consider how to advance to $n+1$ rows by appending row $\mathbf{r}_{n+1}$ to $[\mathbf{X} \quad \mathbf{y}]$:

$$\begin{bmatrix} \mathbf{X} & \mathbf{y} \\ \mathbf{r}_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{r}_{n+1} \end{bmatrix}.$$

The zero rows of $\mathbf{V}$ are discarded.

Multiply the $(p+2) \times (p+1)$ matrix

$$\mathbf{A} = \begin{bmatrix} \hat{\mathbf{U}} & \mathbf{g} \\ \mathbf{0} & q \\ \mathbf{r}_{n+1} \end{bmatrix} \begin{matrix} (p \text{ rows}) \\ (1 \text{ row}) \\ (1 \text{ row}) \end{matrix}$$

by a product of elementary orthogonal matrices, each differing from the identity matrix $\mathbf{I}_{p+2}$ in only two rows and two columns. For $k = 1, 2, ..., p+1$ in turn, the kth orthogonal matrix acts on the kth and last rows to delete the kth element of the last row to alter subsequent elements in the last row. The kth orthogonal matrix has the form

$$\begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & & & s & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & -s & & & & c & \end{bmatrix}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$, and $\theta = \tan^{-1}(a_{p+2,k}/a_{kk})$. After $p+1$ such factors have been applied to matrix $\mathbf{A}$, it will look like

$$\mathbf{A}^* = \begin{bmatrix} \mathbf{U}^* & \mathbf{g}^* \\ \mathbf{0} & q^* \\ \mathbf{0} & 0 \end{bmatrix} \begin{matrix} (p \text{ rows}) \\ (1 \text{ row}) \\ (1 \text{ row}) \end{matrix}$$

where $\mathbf{U}^*$ is also an upper-triangular matrix. You can obtain the solution $\mathbf{b}^{(n+1)}$ to the augmented system of $p+1$ rows by solving

$$\begin{bmatrix} \mathbf{U}^* & \mathbf{g}^* \\ \mathbf{0} & q^* \end{bmatrix} \begin{bmatrix} \mathbf{b}^{(n+1)} \\ -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -q^* \end{bmatrix}.$$

By replacing the last row of $\mathbf{A}^*$ by $\mathbf{r}_{n+2}$ and repeating the factorization, you can continue including additional rows of data in the system. You can add rows indefinitely without increasing the required storage space.

The program below begins with $n = 0$ and $\mathbf{A} = \mathbf{0}$. You enter the rows $\mathbf{r}_m$ successively for $m = 1, 2, ..., p-1$ in turn. You then obtain the current solution $\mathbf{b}$ after entering each subsequent row.

You can also solve weighted least-squares problems and linearly constrained least-squares problems using this program. Make the necessary substitutions described under Orthogonal Factorization earlier in this section.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] [A]	001-42,21,11	Program to input new row.
[STO] 2	002- 44 2	Stores weight in R_2.
1 •	003- 1	
[STO] 1	004- 44 1	Stores $l = 1$ in R_1.
[f] [LBL] 4	005-42,21, 4	
[RCL] [DIM] [A]	006-45,23,11	
[x≷y]	007- 34	
[STO] 0	008- 44 0	Stores $k = p + 2$ in R_0.
[f] [LBL] 5	009-42,21, 5	
[RCL] 1	010- 45 1	
[R/S]	011- 31	
[RCL] 2	012- 45 2	
[×]	013- 20	
[f] [USER] [STO] [A] [f] [USER]	014u 44 11	
[GTO] 5	015- 22 5	
[GTO] 4	016- 22 4	
[f] [LBL] [B]	017-42,21,12	Program to update matrix **A**.
[RCL] [DIM] [A]	018-45,23,11	Recalls dimensions $p + 2$ and $p + 1$.
[x≷y]	019- 34	
[STO] 2	020- 44 2	Stores $p + 2$ in R_2.
[f] [MATRIX] 1	021-42,16, 1	Sets $k = l = 1$.
[f] [LBL] 1	022-42,21, 1	Branch to update ith row.
[g] [CF] 0	023-43, 5, 0	
[RCL] 2	024- 45 2	
[RCL] 0	025- 45 0	
[RCL] [g] [A]	026-45,43,11	Recalls $a_{p+2,k}$.
[RCL] [A]	027- 45 11	Recalls a_{kk}.

Keystrokes	Display	
g TEST 2	028–43,30, 2	Tests $a_{kk} < 0$.
g SF 0	029–43, 4, 0	Sets flag 0 for negative diagonal element.
g ABS	030– 43 16	
g →P	031– 43 1	Calculates θ.
g CLx	032– 43 35	
1	033– 1	
f →R	034– 42 1	Calculates $x = \cos\theta$ and $y = \sin\theta$.
g F? 0	035–43, 6, 0	
CHS	036– 16	Sets $x = c$ and $y = s$.
f I	037– 42 25	Forms $s + ic$.
R↓	038– 33	
f LBL 2	039–42,21, 2	Subroutine to rotate row k.
g R↑	040– 43 33	
RCL A	041– 45 11	Recalls a_{kl}.
RCL 2	042– 45 2	
RCL 1	043– 45 1	
RCL g A	044–45,43,11	Recalls $a_{p+2,l}$.
f I	045– 42 25	Forms $a_{kl} - ia_{p+2,l}$.
×	046– 20	
RCL 2	047– 45 2	
RCL 1	048– 45 1	
STO g A	049–44,43,11	Stores new a_{kl}.
f Re≷Im	050– 42 30	
f USER STO A f USER	051u 44 11	Stores new $a_{p+2,l}$, increments R_0 and R_1.
RCL 1	052– 45 1	Recalls l (column).
RCL 0	053– 45 0	Recalls k (row).
g x≤y	054– 43 10	Tests $k \leq l$.
GTO 2	055– 22 2	Loops back until column reset to 1.
g CF 8	056–43, 5, 8	Turns off Complex mode.
STO 1	057– 44 1	Stores k in R_1 (l).
RCL 2	058– 45 2	

Keystrokes	Display	
g x≤y	059- 43 10	Tests $p+2 \leqslant k$.
g RTN	060- 43 32	Returns at last row.
GTO 1	061- 22 1	Loops back until last row.
f LBL C	062-42,21,13	Program to calculate current solution.
RCL DIM A	063-45,23,11	
ENTER	064- 36	
f DIM A	065-42,23,11	Eliminates last row of **A**.
STO 0	066- 44 0	Stores $p+1$ in R_0.
STO 1	067- 44 1	Stores $p+1$ in R_1.
1	068- 1	
f DIM C	069-42,23,13	Dimensions matrix **C** to $(p+1) \times 1$.
0	070- 0	
STO MATRIX C	071-44,16,13	Sets matrix **C** to **0**.
EEX	072- 26	
9	073- 9	
9	074- 9	
CHS	075- 16	Forms 10^{-99}.
RCL A	076- 45 11	Recalls $q = a_{p+1,p+1}$.
g x=0	077- 43 20	Tests $q = 0$.
R↓	078- 33	Uses 10^{-99} if $q = 0$.
CHS	079- 16	
RCL 0	080- 45 0	
1	081- 1	
STO g C	082-44,43,13	Sets $c_{p+1,1} = -q$.
RCL MATRIX C	083-45,16,13	
RCL MATRIX A	084-45,16,11	
f RESULT C	085-42,26,13	
÷	086- 10	Stores $\mathbf{A}^{-1}\mathbf{C}$ in **C**.
RCL 0	087- 45 0	
1	088- 1	
+	089- 40	
RCL 0	090- 45 0	
f DIM A	091-42,23,11	Dimensions matrix **A** as $(p+2) \times (p+1)$.
1	092- 1	

Keystrokes	Display	
[-]	093- 30	
1	094- 1	
[f] [DIM] [C]	095-42,23,13	Dimensions matrix **C** as $p \times 1$.
[RCL] [A]	096- 45 11	Recalls q.
[f] [MATRIX] 1	097-42,16, 1	Sets $k = l = 1$.
[g] [RTN]	098- 43 32	

Labels used: A, B, C, and 1 through 5.

Registers used: R_0, R_1, and R_2 ($p + 2$ and w).

Matrices used: **A** (working matrix) and **C** (parameter estimates).

Flags used: 0 and 8.

With this program stored, the HP-15C has enough memory to work with up to $p = 4$ parameters. If programs "A" and "C" are deleted, you can work with $p = 5$ parameters. In either case, there is no limit to the number of rows that you can enter.

To use this program:

1. Press 2 [f] [DIM] [(i)] to reserve registers R_0 through R_2.
2. Press [f] [USER] to activate User mode.
3. Enter $(p + 2)$ and $(p + 1)$ into the stack, then press [f] [DIM] [A] to dimension matrix **A**. The dimensions depend on the number of parameters that you use, denoted by p.
4. Press 0 [STO] [MATRIX] [A] to initialize matrix **A**.
5. Enter the weight w_k of the current row, then press [A]. The display should show 1.0000 to indicate that the program is ready for the first row element. (For ordinary least-squares problems, use $w_k = 1$ for each row.)
6. Enter the elements of the row m of matrix **A** by pressing x_{m1} [R/S] x_{m2} [R/S] ... x_{mp} [R/S] y_m [R/S]. After each element is entered, the display should show the number of the next element to be entered. (If you make a mistake while entering the elements, go back and repeat steps 5 and 6 for that row.
7. Press [B] to update the factorization to include the row entered in the previous two steps.

8. Optionally, press [C] [g] [x^2] to calculate and display the residual sum of squares q^2 and to calculate the current solution **b**. Then press [RCL] [C] p times to display b_1, b_2, ..., b_p in turn.
9. Repeat steps 5 through 8 for each additional row.

Example: Use this program and the CPI data from the previous example to fit the model

$$y = b_1 + b_2x_2 + b_3x_3 + r,$$

where y, x_2, and x_3 represent the CPI, PPI, and UR (all as precentages).

This problem involves $p = 3$ parameters, so matrix **A** should be 5×4. The rows of matrix **A** are $(1, x_{m2}, x_{m3}, y_m)$ for $m = 1, 2, ..., 11$. Each row has weight $w_m = 1$.

Keystrokes	Display	
[g] [P/R]		Run mode.
2 [f] [DIM] [(i)]	**2.0000**	Reserves R_0 through R_2.
[f] [USER]	**2.0000**	Activates User mode.
[f] [MATRIX] 0	**2.0000**	Clears matrix memory.
5 [ENTER] 4	**4**	
[f] [DIM] [A]	**4.0000**	Dimensions matrix **A** to 5×4.
0 [STO] [MATRIX] [A]	**0.0000**	Stores zero in all elements.
1 [A]	**1.0000**	Enters weight for row 1.
1 [R/S]	**2.0000**	Enters x_{11}.
3.9 [R/S]	**3.0000**	Enters x_{12}.
3.5 [R/S]	**4.0000**	Enters x_{13}.
5.4 [R/S]	**1.0000**	Enters y_1.
[B]	**5.0000**	Updates factorization.
⋮	⋮	
1 [A]	**1.0000**	Enters weight for row 11.
1 [R/S]	**2.0000**	Enters $x_{11,1}$.
19.3 [R/S]	**3.0000**	Enters $x_{11,2}$.
5.8 [R/S]	**4.0000**	Enters $x_{11,3}$.
11.5 [R/S]	**1.0000**	Enters y_{11}.
[B]	**5.0000**	Updates factorization.

Keystrokes	Display	
C	3.6759	Calculates current estimates and q.
f FIX 9	3.675891055	
g x^2	13.51217505	Calculates residual sum of squares q^2.
RCL C	1.245864306	Displays $b_1^{(11)}$.
RCL C	0.379758235	Displays $b_2^{(11)}$.
RCL C	0.413552221	Displays $b_3^{(11)}$.

These estimates agree (to within 3 in the ninth significant digit) with the results of the preceding example, which uses the normal equations. In addition, you can include additional data and update the parameter estimates. For example, add this data from 1968: CPI = 4.2, PPI = 2.5, and UR = 3.6 .

Keystrokes	Display	
1 A	1.000000000	Enters row weight for new row.
1 R/S	2.000000000	Enters $x_{12,1}$.
2.5 R/S	3.000000000	Enters $x_{12,2}$.
3.6 R/S	4.000000000	Enters $x_{12,3}$.
4.2 R/S	1.000000000	Enters y_{12}.
B	5.000000000	Updates factorization.
C	3.700256908	
g x^2	13.69190119	Calculates residual sum of squares.
RCL C	1.581596327	Displays $b_1^{(12)}$.
RCL C	0.373826487	Displays $b_2^{(12)}$.
RCL C	0.370971848	Displays $b_3^{(12)}$.
f FIX 4	0.3710	
f USER	0.3710	Deactivates User mode.

Eigenvalues of a Symmetric Real Matrix

The eigenvalues of a square matrix **A** are the roots λ_j of its characteristic equation

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0.$$

When **A** is real and symmetric ($\mathbf{A} = \mathbf{A}^T$) its eigenvalues λ_j are all real and possess orthogonal eigenvectors $\mathbf{q}_j$. Then

$$\mathbf{A}\mathbf{q}_j = \lambda_j \mathbf{q}_j$$

and

$$\mathbf{q}_j^T \mathbf{q}_k = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } j = k. \end{cases}$$

The eigenvectors ($\mathbf{q}_1, \mathbf{q}_2, \ldots$) constitute the columns of an orthogonal matrix **Q** which satisfies.

$$\mathbf{Q}^T\mathbf{A}\mathbf{Q} = \text{diag}\,(\lambda_1, \lambda_2, \ldots)$$

and

$$\mathbf{Q}^T = \mathbf{Q}^{-1}.$$

An orthogonal change of variables $\mathbf{x} = \mathbf{Q}\mathbf{z}$, which is equivalent to rotating the coordinate axes, changes the equation of a family of quadratic surfaces ($\mathbf{x}^T\mathbf{A}\mathbf{x} = \text{constant}$) into the form

$$\mathbf{z}^T(\mathbf{Q}^T\mathbf{A}\mathbf{Q})\mathbf{z} = \sum_j^k \lambda_j z_j^2 = \text{constant}.$$

With the equation in this form, you can recognize what kind of surfaces these are (ellipsoids, hyperboloids, paraboloids, cones, cylinders, planes) because the surface's semi-axes lie along the new coordinate axes.

The program below starts with a given matrix **A** that is assumed to be symmetric (if it isn't, it is replaced by $(\mathbf{A} + \mathbf{A}^T)/2$, which *is* symmetric).

Given a symmetric matrix **A**, the program constructs a skew-symmetric matrix (that is, one for which $\mathbf{B} = -\mathbf{B}^T$) using the formula

$$b_{ij} = \begin{cases} \tan(\tfrac{1}{4}\tan^{-1}(2a_{ij}/(a_{ii} - a_{jj}))) & \text{if } i \neq j \text{ and } a_{ij} \neq 0 \\ 0 & \text{if } i = j \text{ or } a_{ij} = 0. \end{cases}$$

Then $\mathbf{Q} = 2(\mathbf{I} + \mathbf{B})^{-1} - \mathbf{I}$ must be an orthogonal matrix whose columns approximate the eigenvalues of **A**; the smaller are all the elements of **B**, the better the approximation. Therefore $\mathbf{Q}^T\mathbf{A}\mathbf{Q}$ must be more nearly diagonal than **A** but with the same eigenvalues. If

$\mathbf{Q}^T\mathbf{AQ}$ is not close enough to diagonal, it is used in place of $\mathbf{A}$ above for a repetition of the process.

In this way, successive orthogonal transformations $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$, ... are applied to $\mathbf{A}$ to produce a sequence $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, ..., where

$$\mathbf{A}_j = (\mathbf{Q}_1\,\mathbf{Q}_2 \ldots \mathbf{Q}_j)^T\mathbf{A}\mathbf{Q}_1\,\mathbf{Q}_2 \ldots \mathbf{Q}_j$$

with each successive $\mathbf{A}_j$ more nearly diagonal than the one before.

This process normally leads to skew matrices whose elements are all small and $\mathbf{A}_j$ rapidly converging to a diagonal matrix $\mathbf{A}$. However, if some of the eigenvalues of matrix $\mathbf{A}$ are very close but far from the others, convergence is slow; fortunately, this situation is rare.

The program stops after each iteration to display

$$\tfrac{1}{2}\sum_j |\text{off-diagonal elements of } \mathbf{A}_j| \,/\, \|\mathbf{A}_j\|_F$$

which measures how nearly diagonal is $\mathbf{A}_j$. If this measure is not negligible, you can press [R/S] to calculate $\mathbf{A}_{j+1}$; if it is negligible, then the diagonal elements of $\mathbf{A}_j$ approximate the eigenvalues of $\mathbf{A}$. The program needs only one iteration for 1×1 and 2×2 matrices, and rarely more than six for 3×3 matrices. For 4×4 matrices the program takes slightly longer and uses all available memory; usually 6 or 7 iterations are sufficient, but if some eigenvalues are very close to each other and relatively far from the rest, then 10 to 16 iterations may be needed.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] [A]	001-42,21,11	
[RCL] [MATRIX] [A]	002-45,16,11	
[STO] [MATRIX] [B]	003-44,16,12	Dimensions B.
[STO] [MATRIX] [C]	004-44,16,13	Dimensions C.
[f] [MATRIX] 4	005-42,16, 4	Transposes A.
[RCL] [MATRIX] [B]	006-45,16,12	
[STO] [RESULT]	007- 44 26	
[+]	008- 40	

Keystrokes	Display	
2	009– 2	
÷	010– 10	
STO MATRIX A	011–44,16,11	Calculates $\mathbf{A} = (\mathbf{A} + \mathbf{A}^T)/2$.
f MATRIX 8	012–42,16, 8	Calculates $\|\mathbf{A}\|_F$.
STO 2	013– 44 2	Stores $\|\mathbf{A}\|_F$ in R_2.
g CLx	014– 43 35	
STO 3	015– 44 3	Initializes off-diagonal sum.
STO MATRIX C	016–44,16,13	Sets $\mathbf{C} = \mathbf{0}$.
f MATRIX 1	017–42,16, 1	Sets $R_0 = R_1 = 1$.
f LBL 0	018–42,21, 0	Routine to construct $\mathbf{Q}$.
RCL 0	019– 45 0	
RCL 1	020– 45 1	
g TEST 5	021–43,30, 5	Tests row = column.
GTO 3	022– 22 3	
g TEST 7	023–43,30, 7	Tests column > row.
GTO 1	024– 22 1	
x≷y	025– 34	
RCL g B	026–45,43,12	
CHS	027– 16	
f USER STO B	028u 44 12	Sets $b_{ij} = -b_{ji}$.
f USER		
GTO 0	029– 22 0	
f LBL 1	030–42,21, 1	Routine for column > row.
RCL g A	031–45,43,11	
g ABS	032– 43 16	Calculates $\|a_{ij}\|$.
STO + 3	033–44,40, 3	Accumulates off-diagonal sum.
g LSTx	034– 43 36	
ENTER	035– 36	
+	036– 40	Calculates $2a_{ij}$.
RCL 0	037– 45 0	
ENTER	038– 36	
RCL g A	039–45,43,11	Recalls a_{ii}.
RCL 1	040– 45 1	
ENTER	041– 36	

Keystrokes	Display	
RCL g A	042-45,43,11	Recalls a_{jj}.
-	043- 30	Calculates $a_{ii} - a_{jj}$.
g TEST 3	044-43,30, 3	Tests $x \geqslant 0$.
GTO 2	045- 22 2	
CHS	046- 16	Keeps angle of rotation between -90° and 90°.
$x \gtrless y$	047- 34	
CHS	048- 16	
$x \gtrless y$	049- 34	
f LBL 2	050-42,21, 2	
g →P	051- 43 1	Calculates angle of rotation.
g CLx	052- 43 35	
4	053- 4	
÷	054- 10	
TAN	055- 25	Calculates b_{ij}.
f USER STO B	056u 44 12	
f USER		
GTO 0	057- 22 0	
f LBL 3	058-42,21, 3	Routine for row = column.
1	059- 1	
STO C	060- 44 13	Sets $c_{ii} = 1$.
f USER STO B	061u 44 12	Sets $b_{ii} = 1$.
f USER		
GTO 0	062- 22 0	
RCL 3	063- 45 3	
RCL ÷ 2	064-45,10, 2	Calculates off-diagonal ratio.
R/S	065- 31	Displays ratio.
2	066- 2	
RCL MATRIX B	067-45,16,12	
÷	068- 10	
RCL MATRIX C	069-45,16,13	
-	070- 30	Calculates $\mathbf{B} = 2(\mathbf{I} + \text{skew})^{-1} - \mathbf{I}$.
RCL MATRIX A	071-45,16,11	
f RESULT C	072-42,26,13	
f MATRIX 5	073-42,16, 5	Calculates $\mathbf{C} = \mathbf{B}^T\mathbf{A}$.

Keystrokes	Display	
RCL MATRIX B	074-45,16,12	
f RESULT A	075-42,26,11	
×	076- 20	Calculates $\mathbf{A} = \mathbf{B}^T\mathbf{AB}$.
GTO A	077- 22 11	

Labels used: A, 0, 1, 2, and 3.

Registers used: R_0, R_1, R_2 (off-diagonal sum), and R_3 ($\|\mathbf{A}_j\|_F$).

Matrices used: **A** ($\mathbf{A}_j$), **B** ($\mathbf{Q}_j$), and **C**.

To use the program:

1. Press 4 f DIM (i) to reserve registers R_0 through R_4.
2. Press f USER to activate User mode.
3. Dimension and enter the elements of matrix **A** using f DIM A and STO A. The dimensions can be up to 4×4, provided that there is sufficient memory available for matrices **B** and **C** having the same dimensions also.
4. Press A to calculate and display the off-diagonal ratio.
5. Press R/S repeatedly until the displayed ratio is negligible, say less than 10^{-8}.
6. Press RCL A repeatedly to observe the elements of matrix **A**. The diagonal elements are the eigenvalues.

Example: What quadratic surface is described by the equation below?

$$\mathbf{x}^T\mathbf{Ax} = [x_1 \quad x_2 \quad x_3] \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= 2x_1x_2 + 4x_1x_3 + 2x_2^2 + 6x_2x_3 + 4x_3^3$$

$$= 7$$

Keystrokes	Display	
g P/R		Run mode.
4 f DIM (i)	4.0000	Allocates memory.
f USER	4.0000	Activates User mode.

Keystrokes	Display		
3 ENTER f DIM A	3.0000		Dimensions **A** to 3×3.
f MATRIX 1	3.0000		Sets R_0 and R_1 to 1.
0 STO A	0.0000		Enters a_{11}.
1 STO A	1.0000		Enters a_{12}.
⋮			
3 STO A	3.0000		Enters a_{32}.
4 STO A	4.0000		Enters a_{33}.
A	0.8660		Calculates ratio—too large.
R/S	0.2304		Again, too large.
R/S	0.1039		Again, too large.
R/S	0.0060		Again, too large.
R/S	3.0463	-05	Again, too large.
R/S	5.8257	-10	Negligible ratio.
RCL A	-0.8730		Recalls $a_{11} = \lambda_1$.
RCL A	-9.0006	-10	Recalls a_{12}.
RCL A	-2.0637	-09	Recalls a_{13}.
RCL A	-9.0006	-10	Recalls a_{21}.
RCL A	9.3429	-11	Recalls $a_{22} = \lambda_2$.
RCL A	1.0725	-09	Recalls a_{23}.
RCL A	-2.0637	-09	Recalls a_{31}.
RCL A	1.0725	-09	Recalls a_{32}.
RCL A	6.8730		Recalls $a_{33} = \lambda_3$.
f USER	6.8730		Deactivates User mode.

In the new coordinate system the equation of the quadratic surface is approximately

$$-0.8730z_1^2 + 0z_2^2 + 6.8730z_3^2 = 7.$$

This is the equation of a hyperbolic cylinder.

Eigenvectors of a Symmetric Real Matrix

As discussed in the previous application, a real symmetric matrix **A** has real eigenvalues λ_1, λ_2, ... and corresponding orthogonal eigenvectors $\mathbf{q}_1$, $\mathbf{q}_2$, ...

This program uses inverse iteration to calculate an eigenvector $\mathbf{q}_k$ that corresponds to the eigenvalue λ_k such that $\|\mathbf{q}_k\|_R = 1$. The technique uses an initial vector $\mathbf{z}^{(0)}$ to calculate subsequent vectors $\mathbf{w}^{(n)}$ and $\mathbf{z}^{(n)}$ repeatedly from the equations

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{w}^{(n+1)} = \mathbf{z}^{(n)}$$
$$\mathbf{z}^{(n+1)} = s\,\mathbf{w}^{(n+1)}/\|\mathbf{w}^{(n+1)}\|_R$$

where s denotes the sign of the first component of $\mathbf{w}^{(n+1)}$ having the largest absolute value. The iterations continue until $\mathbf{z}^{(n)}$ converges. That vector is an eigenvector $\mathbf{q}_k$ corresponding to the eigenvalue λ_k.

The value used for λ_k need not be exact; the calculated eigenvector is determined accurately in spite of small inaccuracies in λ_k. Furthermore, don't be concerned about having too accurate an approximation to λ_k; the HP-15C can calculate the eigenvector even when $\mathbf{A} - \lambda_k\mathbf{I}$ is very ill-conditioned.

This technique requires that vector $\mathbf{z}^{(0)}$ have a nonzero component along the unknown eigenvector $\mathbf{q}_k$. Because there are no other restrictions on $\mathbf{z}^{(0)}$, the program uses random components for $\mathbf{z}^{(0)}$. At the end of each iteration, the program displays $\|\mathbf{z}^{(n+1)} - \mathbf{z}^{(n)}\|_R$ to show the rate of convergence.

This program can accommodate a matrix $\mathbf{A}$ that isn't symmetric but has a diagonal Jordan canonical form—that is, there exists some nonsingular matrix $\mathbf{P}$ such that $\mathbf{P}^{-1}\mathbf{AP} = \text{diag}(\lambda_1,\lambda_2,...)$.

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000-	
f LBL C	001-42,21,13	
STO 2	002- 44 2	Stores eigenvalue in R_2.
RCL MATRIX A	003-45,16,11	
STO MATRIX B	004-44,16,12	Stores **A** in **B**.
RCL DIM A	005-45,23,11	
STO 0	006- 44 0	
f LBL 4	007-42,21, 4	
RCL 0	008- 45 0	
STO 1	009- 44 1	
RCL B	010- 45 12	

Keystrokes	Display	
RCL - 2	011-45,30, 2	
STO B	012- 44 12	Modifies diagonal elements of **B**.
f DSE 0	013-42, 5, 0	
GTO 4	014- 22 4	
RCL DIM A	015-45,23,11	
1	016- 1	
f DIM C	017-42,23,13	Dimensions **C** to $n \times 1$.
f MATRIX 1	018-42,16, 1	
f LBL 5	019-42,21, 5	
f RAN#	020- 42 36	
f USER STO C f USER	021u 44 13	Stores random components in **C**.
GTO 5	022- 22 5	
f LBL 6	023-42,21, 6	Routine for iterating $\mathbf{z}^{(n)}$ and $\mathbf{w}^{(n)}$.
RCL MATRIX C	024-45,16,13	
STO MATRIX D	025-44,16,14	Stores $\mathbf{z}^{(n)}$ in **D**.
STO RESULT	026- 44 26	
RCL MATRIX B	027-45,16,12	
÷	028- 10	Calculates $\mathbf{w}^{(n+1)}$ in **C**.
ENTER	029- 36	
f MATRIX 7	030-42,16, 7	
÷	031- 10	Calculates $\pm\mathbf{z}^{(n+1)}$ in **C**.
f MATRIX 1	032-42,16, 1	
f LBL 7	033-42,21, 7	Routine to find sign of largest element.
f USER RCL C f USER	034u 45 13	
ENTER	035- 36	(This line skipped for last element.)
g ABS	036- 43 16	
1	037- 1	
g TEST 6	038-43,30, 6	Tests $\lvert a_j \rvert \neq 1$.
GTO 7	039- 22 7	
RCL MATRIX C	040-45,16,13	
g LSTx	041- 43 36	Recalls extreme a_j.
÷	042- 10	Calculates $\mathbf{z}^{(n+1)}$ in **C**.

Keystrokes	Display	
RCL MATRIX D	043-45,16,14	
STO RESULT	044- 44 26	
-	045- 30	Calculates $\mathbf{z}^{(n+1)} - \mathbf{z}^{(n)}$ in **D**.
f MATRIX 7	046-42,16, 7	Calculates $\|\mathbf{z}^{(n+1)} - \mathbf{z}^{(n)}\|_R$.
f MATRIX 1	047-42,16, 1	Sets $R_0 = R_1 = 1$ for viewing **C**.
R/S	048- 31	Displays convergence parameter.
GTO 6	049- 22 6	

Labels used: C, 4, 5, 6, and 7.

Registers used: R_0, R_1, and R_2 (eigenvalue).

Matrices used: **A** (original matrix), **B** (**A** − λ**I**), **C** ($\mathbf{z}^{(n+1)}$), and **D** ($\mathbf{z}^{(n+1)} - \mathbf{z}^{(n)}$).

To use this program:

1. Press 2 f DIM (i) to reserve registers R_0, R_1, and R_2.
2. Press f USER to activate User mode.
3. Dimension and enter the elements into matrix **A** using f DIM A, f MATRIX 1, and STO A.
4. Key in the eigenvalue and press C. The display shows the correction parameter $\|\mathbf{z}^{(1)} - \mathbf{z}^{(0)}\|_R$.
5. Press R/S repeatedly until the correction parameter is negligibly small.
6. Press RCL C repeatedly to view the components of $\mathbf{q}_k$, the eigenvector.

Example: For matrix **A** of the previous example,

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

calculate the eigenvectors $\mathbf{q}_1$, $\mathbf{q}_2$, and $\mathbf{q}_3$.

Keystrokes	Display		
g P/R			Run mode.
2 f DIM (i)	2.0000		Reserves registers R_0 through R_2.
f USER	2.0000		Activates User mode.
3 ENTER f DIM A	3.0000		Dimensions matrix **A** to 3×3.
f MATRIX 1	3.0000		
0 STO A	0.0000		Enters elements of **A**.
1 STO A	1.0000		
⋮			
4 STO A	4.0000		
.8730 CHS	-0.8730		Enters $\lambda_1 = -0.8730$ (approximation).
C	0.8982		$\|\mathbf{z}^{(1)} - \mathbf{z}^{(0)}\|$.*
R/S	0.0001		$\|\mathbf{z}^{(2)} - \mathbf{z}^{(1)}\|$.*
R/S	2.4000	-09	$\|\mathbf{z}^{(3)} - \mathbf{z}^{(2)}\|$.*
R/S	1.0000	-10	$\|\mathbf{z}^{(4)} - \mathbf{z}^{(3)}\|$.*
R/S	0.0000		$\|\mathbf{z}^{(5)} - \mathbf{z}^{(4)}\|$.*
RCL C	1.0000		Eigenvector for λ_1.
RCL C	0.2254		
RCL C	-0.5492		
0 C	0.8485		Uses $\lambda_2 = 0$ (approximation).
R/S	0.0000		
RCL C	-0.5000		Eigenvector for λ_2.
RCL C	1.0000		
RCL C	-0.5000		
6.8730 C	0.7371		Uses $\lambda_3 = 6.8730$ (approximation).
R/S	1.9372	-06	
R/S	1.0000	-10	
R/S	0.0000		

*The correction norms will vary, depending upon the current random number seed.

Keystrokes	Display	
RCL C	0.3923	} Eigenvector for λ_3.
RCL C	0.6961	
RCL C	1.0000	
f USER	1.0000	Deactivates User mode.

If matrix **A** is no larger than 3×3, this program can be included with the previous eigenvalue program. Since the eigenvalue program modifies matrix **A**, the original eigenvalues must be saved and the original matrix reentered in matrix **A** before running the eigenvector program. The following program can be added to store the calculated eigenvalues in matrix **E**.

Keystrokes	Display	
f LBL E	127-42,21,15	
RCL DIM A	128-45,23,11	
STO 0	129- 44 0	
1	130- 1	
f DIM E	131-42,23,15	Dimensions **E** to $n \times 1$.
f LBL 8	132-42,21, 8	
RCL 0	133- 45 0	
ENTER	134- 36	
RCL g A	135-45,43,11	Recalls diagonal element.
RCL 0	136- 45 0	
1	137- 1	
STO g E	138-44,43,15	Stores a_{ii} in e_i.
f DSE 0	139-42, 5, 0	
GTO 8	140- 22 8	
f MATRIX 1	141-42,16, 1	Resets $R_0 = R_1 = 1$.
g RTN	142- 43 32	
g P/R		Run mode.

Labels used: E and 8.

Registers used: no additional registers.

Matrices used: **A** (from previous program) and **E** (eigenvalues).

To use the combined eigenvalue, eigenvalue storage, and eigenvector programs for an **A** matrix up to 3×3:

1. Execute the eigenvalue program as described earlier.

2. Press [E] to store the eigenvalues.
3. Enter again the elements of the original matrix into **A**.
4. Recall the desired eigenvalue from matrix **E** using [RCL] [E].
5. Execute the eigenvector program as described above.
6. Repeat steps 4 and 5 for each eigenvalue.

Optimization

Optimization describes a class of problems in which the object is to find the minimum or maximum value of a specified function. Often, the interest is focused on the behavior of the function in a particular region.

The following program uses the method of steepest descent to determine local minimums or maximums for a real-valued function of two or more variables. This method is an iterative procedure that uses the gradient of the function to determine successive sample points. Four input parameters control the sampling plan.

For the function

$$f(\mathbf{x}) = f(x_1, x_2, ..., x_n)$$

the gradient of f, ∇f, is defined by

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}.$$

The critical points of $f(\mathbf{x})$ are the solutions to $\nabla f(\mathbf{x}) = \mathbf{0}$. A critical point may be a local minimum, a local maximum, or a point that is neither.

The gradient of $f(\mathbf{x})$ evaluated at a point $\mathbf{x}$ gives the direction of steepest ascent—that is, the way in which $\mathbf{x}$ should be changed in order to cause the most rapid increase in $f(\mathbf{x})$. The negative gradient gives the direction of steepest descent. The direction vector is

$$\mathbf{s} = \begin{cases} -\nabla f(\mathbf{x}) & \text{for finding a minimum} \\ \nabla f(\mathbf{x}) & \text{for finding a maximum.} \end{cases}$$

Once the direction is determined from the gradient, the program looks for the optimum distance to move from $\mathbf{x}_j$ in the direction indicated by $\mathbf{s}_j$—the distance that gives the greatest improvement in $f(\mathbf{x})$ toward a minimum or maximum.

To do this, the program finds the optimum value t_j by calculating the slope of the function

$$g_j(t) = f(\mathbf{x}_j + t\mathbf{s}_j)$$

at increasing values of t until the slope changes sign. This procedure is called "bounding search" since the program tries to bound the desired value t_j within an interval. When the program finds a change of sign, it then reduces the interval by halving it $j + 1$ times to find the best t value near $t = 0$. This procedure is called "interval reduction"—it yields more accurate values for t_j as $\mathbf{x}_j$ converges toward the desired solution. (These two processes are collectively called "line search.") The new value of $\mathbf{x}$ is then

$$\mathbf{x}_{j+1} = \mathbf{x}_j + t_j\mathbf{s}_j.$$

The program uses four parameters that define how it proceeds toward the desired solution. Although no method of line search can guarantee success for finding an optimum value of t, the first two parameters give you considerable flexibility in specifying how the program samples t.

d Determines the initial step u_1 for the bounding search. The first value of t tried is

$$u_1 = \frac{d}{(j+1)\|\mathbf{s}_j\|_F} .$$

This corresponds to a distance of

$$\|(\mathbf{x}_j + u_1\mathbf{s}_j) - \mathbf{x}_j\|_F = \frac{d}{j+1} ,$$

which shows that d and the iteration number define how close to the last $\mathbf{x}$ value the program starts the bounding search.

a Determines the values u_2, u_3, ... of subsequent steps in the bounding search. These values of t are defined by

$$u_{i+1} = au_i.$$

Essentially, a is an expansion factor that is normally greater than 1, producing an increasing sequence of values of t.

e Determines the acceptable tolerance on the size of the gradient. The iterative process stops when

$$\|\nabla f(\mathbf{x}_j)\|_F \leq e.$$

N Determines the maximum number of iterations that the program will attempt in each of two procedures: the bounding search and the overall optimization procedure. That is, the program halts if the bounding search finds no change of sign within N iterations. Also, the program halts if the norm of the gradient is still too large at $\mathbf{x}_N$. Each of these situations results in an **Error 1** display. (They can be distinguished by pressing [←].) You can continue running the program if you desire.

The program requires that you enter a subroutine that evaluates $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$. This subroutine must be labeled "E", use the vector $\mathbf{x}$ stored in matrix **A**, return the gradient in matrix **E**, and place $f(\mathbf{x})$ in the X-register.

In addition, the program requires an initial estimate $\mathbf{x}_0$ of the desired critical point. This vector must be stored in matrix **A**.

The program has the following characteristics:

- The program searches for any point $\mathbf{x}$ where $\nabla f(\mathbf{x}) = \mathbf{0}$. Nothing prevents convergence to a saddle-point, for example. In general, you must use other means to determine the nature of the critical point that is found. (Also, this program does not address the problem of locating a maximum or minimum on the boundary of the domain of $f(\mathbf{x})$.)
- You may adjust the convergence parameters after starting the program. In many cases, this dramatically reduces the time necessary for convergence. Here are some helpful hints:
  - If the program consistently enters the interval reduction phase after sampling only one point u_1, the initial step size may be too large. Try reducing the magnitude of d to produce a more efficient search.
  - If the results of the bounding search look promising (that is, the slopes are decreasing in magnitude), but then begin to increase in magnitude, the search may have skipped past a critical point. Try reducing a to produce more close sampling; you may have to increase N also.

- You can replace [R/S] at line 102 with [PSE] or perhaps delete it entirely if you have no interest in the intermediate results.
- For a function of n variables, the program requires $4n + 1$ registers devoted to matrices.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] 8	001-42,21, 8	Routine to swap **A** and **C** using **E**.
[RCL] [MATRIX] [C]	002-45,16,13	
[STO] [MATRIX] [E]	003-44,16,15	
[RCL] [MATRIX] [A]	004-45,16,11	
[STO] [MATRIX] [C]	005-44,16,13	
[RCL] [MATRIX] [E]	006-45,16,15	
[STO] [MATRIX] [A]	007-44,16,11	
[g] [RTN]	008- 43 32	
[f] [LBL] 7	009-42,21, 7	Line search routine.
[RCL] 4	010- 45 4	
[RCL] [÷] 6	011-45,10, 6	
[STO] 8	012- 44 8	Stores $d/(j+1)$ in R_8.
[GSB] [E]	013- 32 15	
[RCL] [MATRIX] [E]	014-45,16,15	
[STO] [MATRIX] [D]	015-44,16,14	
[RCL] [MATRIX] [D]	016-45,16,14	
[g] [F?] 0	017-43, 6, 0	
[CHS]	018- 16	For minimum, changes sign of gradient.
[f] [MATRIX] 8	019-42,16, 8	Calculates $\|\nabla f(\mathbf{x})\|$.
[g] [x=0]	020- 43 20	
[g] [RTN]	021- 43 32	Exits if $\|\nabla f(\mathbf{x})\| = 0$.
[1/x]	022- 15	
[RCL] [×] 8	023-45,20, 8	Calculates u_1.
[STO] .1	024- 44 .1	Stores u_1 in $R_{.1}$.
0	025- 0	
[STO] .0	026- 44 .0	
[RCL] 5	027- 45 5	
[STO] 7	028- 44 7	Stores counter in R_7.

Keystrokes	Display	
f LBL 6	029-42,21, 6	Bounding search begins.
RCL .1	030- 45 .1	
GSB 3	031- 32 3	
f PSE	032- 42 31	Shows slope.
g F? 0	033-43, 6, 0	
CHS	034- 16	
g TEST 4	035-43,30, 4	Tests for slope change.
GTO 5	036- 22 5	Branch to interval reduction.
GSB 8	037- 32 8	Restores original matrix to **A**.
RCL .1	038- 45 .1	
STO .0	039- 44 .0	Stores u_i in $R_{.0}$.
RCL 2	040- 45 2	
STO × .1	041-44,20, .1	Stores u_{i+1} in $R_{.1}$.
f DSE 7	042-42, 5, 7	Decrements counter.
GTO 6	043- 22 6	Branch to continue.
RCL MATRIX A	044-45,16,11	
g ABS	045- 43 16	Displays **Error 1** with **A** in X-register.
GTO 6	046- 22 6	Branch for continuation.
f LBL 5	047-42,21, 5	Interval reduction routine.
RCL 6	048- 45 6	
STO 7	049- 44 7	Stores $j + 1$ in R_7.
f LBL 4	050-42,21, 4	
GSB 8	051- 32 8	Restores original matrix to **A**.
RCL .0	052- 45 .0	
RCL + .1	053-45,40, .1	
2	054- 2	
÷	055- 10	
STO 8	056- 44 8	Calculates midpoint of interval.
GSB 3	057- 32 3	Calculates slope.
g F? 0	058-43, 6, 0	
CHS	059- 16	Changes sign for minimum.

Keystrokes	Display	
1	060– 1	
1	061– 1	
STO I	062– 44 25	Stores interval register number.
R↓	063– 33	
g TEST 1	064–43,30, 1	
f DSE I	065–42, 5,25	
RCL 8	066– 45 8	
STO (i)	067– 44 24	Stores midpoint in $R_{.0}$ or $R_{.1}$.
f DSE 7	068–42, 5, 7	Decrements counter.
GTO 4	069– 22 4	
g RTN	070– 43 32	Exits when counter is zero.
f LBL 3	071–42,21, 3	Routine to calculate slope.
RCL MATRIX D	072–45,16,14	
f RESULT C	073–42,26,13	
×	074– 20	
RCL MATRIX A	075–45,16,11	
+	076– 40	Calculates point $\mathbf{x}_j + t\mathbf{s}_j$.
GSB 8	077– 32 8	Swaps original matrix and new point.
GSB E	078– 32 15	Calculates $\nabla f(\mathbf{x})$ in **E**.
STO 9	079– 44 9	Stores $f(\mathbf{x})$ in R_9.
RCL MATRIX E	080–45,16,15	
RCL MATRIX D	081–45,16,14	
f RESULT B	082–42,26,12	
f MATRIX 5	083–42,16, 5	Calculates slope as $(\nabla f)^T\mathbf{s}$.
1	084– 1	
ENTER	085– 36	
RCL g B	086–45,43,12	
g RTN	087– 43 32	Exits with slope in X-register.
f LBL A	088–42,21,11	Main routine.
0	089– 0	
STO 6	090– 44 6	
f LBL 2	091–42,21, 2	
1	092– 1	

Keystrokes	Display	
STO + 6	093-44,40, 6	Stores $j+1$ in R_6.
f SCI 3	094-42, 8, 3	
GSB 7	095- 32 7	Branches to line search.
RCL 6	096- 45 6	
f FIX 0	097-42, 7, 0	
f PSE	098- 42 31	Pauses with $j+1$ in display.
f MATRIX 1	099-42,16, 1	Sets $R_0 = R_1 = 1$ for viewing.
f SCI 3	100-42, 8, 3	
RCL 9	101- 45 9	Recalls $f(\mathbf{x})$.
R/S	102- 31	Stops program.
RCL 3	103- 45 3	Recalls e.
RCL MATRIX E	104-45,16,15	
f MATRIX 8	105-42,16, 8	Calculates $\|\nabla f(\mathbf{x})\|$.
g x≤y	106- 43 10	Tests $\|\nabla f(\mathbf{x})\| \leq e$.
GTO B	107- 22 12	Branch for showing solution.
f PSE	108- 42 31	Shows $\|\nabla f(\mathbf{x})\|$.
RCL 5	109- 45 5	
RCL 6	110- 45 6	
g TEST 8	111-43,30, 8	Tests $(j+1) < N$.
GTO 2	112- 22 2	Branch to continue iterating.
RCL MATRIX C	113-45,16,13	
g ABS	114- 43 16	Displays **Error 1** with C in X-register.
GTO 2	115- 22 2	Branch for continuing.
f LBL B	116-42,21,12	Routine to show solution.
g SF 9	117-43, 4, 9	Sets blink flag.
R/S	118- 31	Stops with $\|\nabla f(\mathbf{x}_{j+1})\|$ in display.
GTO B	119- 22 12	Looping branch.

Labels used: A, B, and 2 through 8.

Registers used: R_2 through R_9, $R_{.0}$, $R_{.1}$, and Index register.

Matrices used: A, B, C, D, and E.

Your subroutine, labeled "E", may use any labels and registers not listed above, plus the Index register, matrix B, and matrix E (which should contain your calculated gradient).

To use the program:

1. Enter your subroutine into program memory.
2. Press 11 [f] [DIM] [(i)] to reserve registers R_0 through $R_{.1}$. (Your subroutine may require additional registers.)
3. Set flag 0 if you're seeking a local minimum; clear flag 0 if you're seeking a local maximum.
4. Dimension matrix **A** to $n \times 1$, where n is the number of variables.
5. Store the required data in memory:
   - Store the initial estimate $\mathbf{x}_0$ in matrix **A**.
   - Store a in R_2.
   - Store e in R_3.
   - Store d in R_4.
   - Store N in R_5.
6. Press [GSB] [A] to view the slopes during the iteration procedure.
   - View the iteration number and the value of $f(\mathbf{x})$.
   - If **Error 1** appears, press [←] to clear the message. Then either go back to step 5 and possibly revise parameters as needed, or press [←] [R/S] to provide one more bounding search iteration or one more optimization iteration. (If the descriptor of matrix **A** was in the display when the error occurred, the number of bounding search iterations exceeded N; if the descriptor of matrix **C** was in the display, the number of optimization iterations exceeded N.)
7. Press [R/S] to view the norm of the gradient and to start the next iteration.
   - If the display flashes the norm of the gradient, press [←] and then recall the values of **x** in matrix **A**.

- If the iteration number and value of $f(\mathbf{x})$ are displayed, repeat this step as often as necessary to obtain the solution or go back to step 5 and revise parameters as needed.

Example: Use the optimization program to find the dimensions of the box of largest volume with the sum of the length and girth (perimeter of cross section) equaling 100 centimeters.

For this problem

$$l + (2h + 2w) = 100$$

$$v = whl$$

$$v(w,h) = wh(100 - 2h - 2w)$$

$$= 100wh - 2wh^2 - 2hw^2$$

$$\nabla v(w,h) = \begin{bmatrix} 2h(50 - h - 2w) \\ 2w(50 - w - 2h) \end{bmatrix}.$$

The solution should satisfy $w + h < 50$, $w > 0$, and $h > 0$.

First, enter a subroutine to calculate the gradient and the volume.

Keystrokes	Display	
f LBL E	120-42,21,15	Function subroutine.
RCL DIM A	121-45,23,11	
f DIM E	122-42,23,15	
f MATRIX 1	123-42,16, 1	
f USER RCL A	124u 45 11	
f USER		
STO .2	125- 44 .2	Stores w in $R_{.2}$.
STO E	126- 44 15	Stores w in e_2.
RCL A	127- 45 11	
STO .3	128- 44 .3	Stores h in $R_{.3}$.
f MATRIX 1	129-42,16, 1	
STO E	130- 44 15	Stores h in e_1.
+	131- 40	
5	132- 5	
0	133- 0	
-	134- 30	

Keystrokes	Display	
CHS	135- 16	
2	136- 2	
×	137- 20	Calculates $l = 2(50 - h - w)$.
f x≷ .2	138-42, 4, .2	Stores l in $R_{.2}$.
STO × .3	139-44,20, .3	Stores wh in $R_{.3}$.
RCL .2	140- 45 .2	
RCL MATRIX E	141-45,16,15	
f RESULT E	142-42,26,15	
×	143- 20	
RCL .3	144- 45 .3	
RCL + .3	145-45,40, .3	
−	146- 30	Replaces e_i with $le_i - 2wh$, the gradient elements.
RCL .2	147- 45 .2	
RCL × .3	148-45,20, .3	Calculates lwh.
g RTN	149- 43 32	

Now enter the necessary information and run the program.

Keystrokes	Display	
g P/R		Run mode.
13 f DIM (i)	13.0000	Reserves R_0 through $R_{.3}$.
g CF 0	13.0000	Finds local maximum.
f USER	13.0000	Activates User mode.
f MATRIX 1	13.0000	
2 ENTER 1	1	Enters dimensions for matrix **A**.
f DIM A	1.0000	Dimensions matrix **A** to 2×1.
15 STO A	15.0000	
STO A	15.0000	Stores initial estimate: $l = w = 15$.
3 STO 2	3.0000	Stores $a = 3$.
0.1 STO 3	0.1000	Stores $e = 0.1$.
0.05 STO 4	0.0500	Stores $d = 0.05$.

Keystrokes	Display		
4 [STO] 5	4.0000		Stores $N = 4$.
[A]	4.415	04	Slope at u_1.
	4.243	04	Slope at u_2.
	3.718	04	Slope at u_3.
	2.045	04	Slope at u_4.
	Error 1		
[←]	A 2 1		Bounding search failed.

Since the results so far look promising (the derivatives are decreasing in magnitude), allow five additional samples in this bounding search and set $N = 8$ for all subsequent iterations.

Keystrokes	Display		
5 [STO] 7	5.000	00	Sets counter to 5.
8 [STO] 5	8.000	00	Sets N to 8.
[R/S]	-3.849	04	Slope at u_5 (sign change).
	1.		$j + 1$.
	9.253	03	Volume at this iteration.
[R/S]	3.480	01	Gradient.
	1.121	03	Slope at u_1.
	9.431	02	Slope at u_2.
	4.126	02	Slope at u_3.
	-1.139	03	Slope at u_4 (sign change).
	2.		$j + 1$.
	9.259	03	Volume at this iteration.
[R/S]	5.479	-01	Gradient.
	-6.127	-01	Slope at u_1 (sign change).
	3.		$j + 1$.
	9.259	03	Volume at this iteration.
[R/S]	7.726	-02	Gradient less than e.
[←]	7.726	-02	Stops blinking.
[f] [FIX] 4	0.0773		
[RCL] [A]	16.6661		Recalls h from a_1.
[RCL] [A]	16.6661		Recalls w from a_2.

Keystrokes	Display	
f USER	**16.6661**	
f MATRIX 0	**16.6661**	Deallocates matrix memory.

The desired box size is $16.6661 \times 16.6661 \times 33.3355$ centimeters. (An alternate method of solving this problem would be to solve the linear system represented by $\nabla v(w,h) = \mathbf{0}$.)