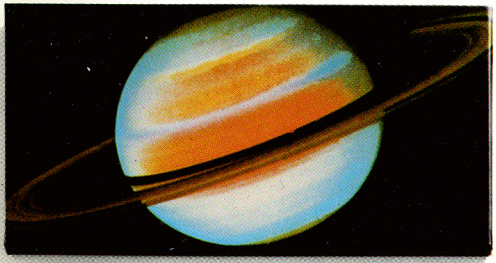
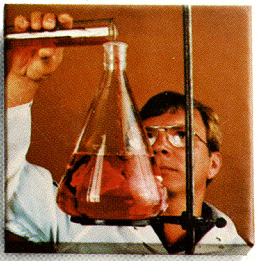
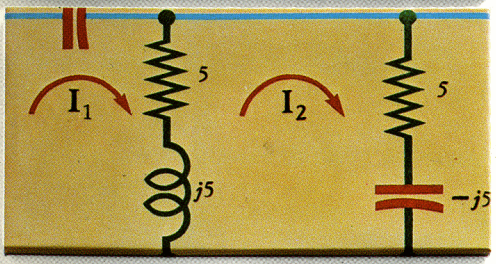
HEWLETT-PACKARD

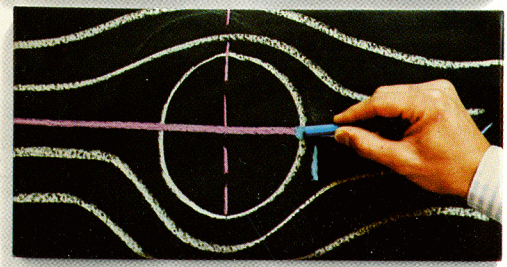
HP-15C

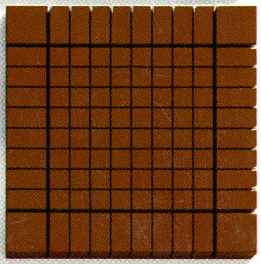
ADVANCED FUNCTIONS HANDBOOK











NOTICE

Hewlett-Packard Company makes no express or implied warranty with regard to the keystroke procedures and program material offered or their merchantability or their fitness for any particular purpose. The keystroke procedures and program material are made available solely on an "as is" basis, and the entire risk as to their quality and performance is with the user. Should the keystroke procedures or program material prove defective, the user (and not Hewlett-Packard Company nor any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Hewlett-Packard Company shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of the keystroke procedures or program material.

HP-15C

Advanced Functions Handbook

August 1982

00015-90011

Printed in U.S.A.

Contents

Introduction

The HP-15C provides several advanced capabilities never before combined so conveniently in a handheld calculator:

- Finding the roots of equations.
- Evaluating definite integrals.
- Calculating with complex numbers.
- Calculating with matrices.

The *HP-15C Owner's Handbook* gives the basic information about performing these advanced operations. It also includes numerous examples that show how to use these features. The owner's handbook is your primary reference for information about the advanced functions.

This *HP-15C Advanced Functions Handbook* continues where the owner's handbook leaves off. In this handbook you will find information about how the HP-15C performs the advanced computations and information that explains how to interpret the results that you get.

This handbook also contains numerous programs, or applications. These programs serve two purposes. First, they suggest ways of using the advanced functions, so that you might use these capabilities more effectively in your own applications. Second, the programs cover a wide range of applications—they may be useful to you in the form presented in this handbook.

> **Note:** The discussions of most topics in this handbook presume that you already understand the basic information about using the advanced functions and that you are generally familiar with the subject matter being discussed.

Section 1

Using [SOLVE] Effectively

The [SOLVE] algorithm provides an effective method for finding a root of an equation. This section describes the numerical method used by [SOLVE] and gives practical information about using [SOLVE] in various situations.

Finding Roots

In general, no numerical technique can be guaranteed to find a root of every equation that has one. Because a finite number of digits are used, the calculated function may differ from the theoretical function in certain intervals of x, it may not be possible to represent the roots exactly, or it may be impossible to distinguish between zeros and discontinuities of the function being used. Because the function can be sampled at only a finite number of places, it's also possible to conclude falsely that the equation has no roots.

Despite these inherent limitations on any numerical method for finding roots, an effective method—like that used by [SOLVE]—should strive to meet each of the following objectives:

- If a real root exists and can be exactly represented by the calculator, it should be returned. Note that the calculated function may underflow (and be set to zero) for some values of x other than the true roots.
- If a real root exists, but it can't be exactly represented by the calculator, the value returned should differ from the true root only in the last significant digit.
- If no real root exists, an error message should be displayed.

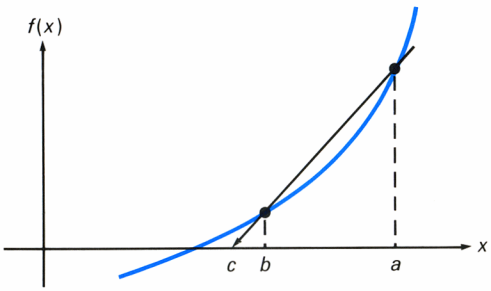
The [SOLVE] algorithm was designed with these objectives in mind. It is also easy to use and requires little of the calculator's memory. And because [SOLVE] in a program can detect the situation of not finding a root, your programs can remain entirely automatic regardless of whether [SOLVE] finds a root.

How [SOLVE] Samples

The [SOLVE] routine uses only five registers of allocatable memory in the HP-15C. The five registers hold three sample values (a, b, and c) and two previous function values ($f(a)$ and $f(b)$) while your function subroutine calculates $f(c)$.

The key to the effectiveness of [SOLVE] is how the next sample value c is found.

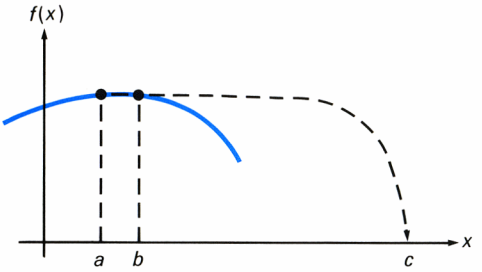
Normally, [SOLVE] uses the secant method to select the next value. This method uses the values of a, b, $f(a)$, and $f(b)$ to predict a value c where $f(c)$ might be close to zero.



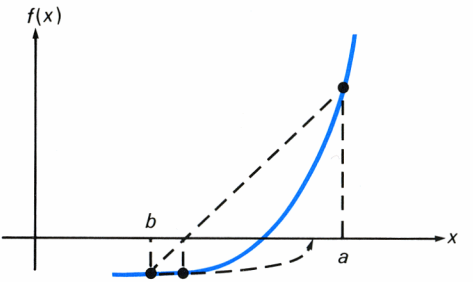
If c isn't a root, but $f(c)$ is closer to zero than $f(b)$, then b is relabeled as a, c is relabeled as b, and the prediction process is repeated. Provided the graph of $f(x)$ is smooth and provided the initial values of a and b are close to a simple root, the secant method rapidly converges to a root.

However, under certain conditions the secant method doesn't suggest a next value that will bound the search or move the search closer to a root, such as finding a sign change or a smaller function magnitude. In such cases, [SOLVE] uses a different approach.

If the calculated secant is nearly horizontal, [SOLVE] modifies the secant method to ensure that $|c - b| \leq 100|a - b|$. This is especially important because it also reduces the tendancy for the secant method to go astray when rounding error becomes significant near a root.



If SOLVE has already found values a and b such that $f(a)$ and $f(b)$ have opposite signs, it modifies the secant method to ensure that c always lies within the interval containing the sign change. This guarantees that the search interval decreases with each iteration, eventually finding a root.



If SOLVE hasn't found a sign change and a sample value c doesn't yield a function value with diminished magnitude, then SOLVE fits a parabola through the function values at a, b, and c. SOLVE finds the value d at which the parabola has its maximum or minimum, relabels d as a, and then continues the search using the secant method.

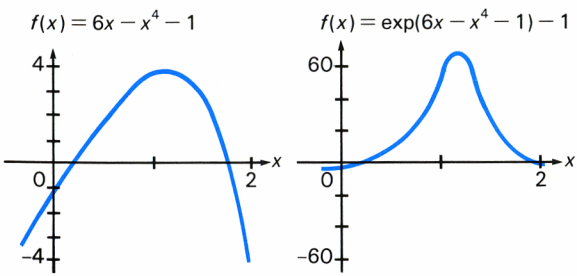
SOLVE abandons the search for a root only when three successive parabolic fits yield no decrease in the function magnitude or when $d = b$. Under these conditions, the calculator displays **Error 8**. Because b represents the point with the smallest sampled function magnitude, b and $f(b)$ are returned in the X- and Z-registers, respectively. The Y-register contains the value of a or c. With this information, you can decide what to do next. You might resume the search where it left off, or direct the search elsewhere, or decide that $f(b)$ is negligible so that $x = b$ is a root, or transform the equation into another equation easier to solve, or conclude that no root exists.

Handling Troublesome Situations

The following information is useful for working with problems that could yield misleading results. Inaccurate roots are caused by calculated function values that differ from the intended function values. You can frequently avoid trouble by knowing how to diagnose inaccuracy and reduce it.

Easy Versus Hard Equations

The two equations $f(x) = 0$ and $e^{f(x)} - 1 = 0$ have the same real roots, yet one is almost always much easier to solve numerically than the other. For instance, when $f(x) = 6x - x^4 - 1$, the first equation is easier. When $f(x) = \ln(6x - x^4)$, the second is easier. The difference lies in how the function's graph behaves, particularly in the vicinity of a root.



In general, every equation is one of an infinite family of equivalent equations with the same real roots. And some of those equations must be easier to solve than others. While [SOLVE] may fail to find a root for one of those equations, it may succeed with another.

Inaccurate Equations

[SOLVE] can't calculate an equation's root incorrectly *unless the function is incorrectly calculated.* The accuracy of your function subroutine affects the accuracy of the root that you find.

You should be aware of conditions that might cause your calculated function value to differ from the theoretical value you want it to have. [SOLVE] can't infer intended values of your function. Frequently, you can minimize calculation error by carefully writing your function subroutine.

Equations With Several Roots

The task of finding all roots of an equation becomes more difficult as the number of roots increases. And any roots that cluster closely will usually defy attempts at accurate resolution. You can use *deflation* to eliminate roots, as described in the *HP-15C Owner's Handbook.*

An equation with a multiple root is characterized by the function and its first few higher-order derivatives being zero at the multiple root. When [SOLVE] finds a double root, the last half of its digits may be inaccurate. For a triple root, two-thirds of the root's digits tend to be obscured. A quadruple root tends to lose about three-fourths of its digits.

Using [SOLVE] With Polynomials

Polynomials are among the easiest functions to evaluate. That is why they are traditionally used to approximate functions that model physical processes or more complex mathematical functions.

A polynomial of degree n can be represented as

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \ .$$

This function equals zero at no more than n real values of x, called zeros of the polynomial. A limit to the number of *positive* zeros of this function can be determined by counting the number of times

the signs of the coefficients change as you scan the polynomial from left to right. Similarly, a limit to the number of *negative* zeros can be determined by scanning a new function obtained by substituting $-x$ in place of x in the original polynomial. If the actual number of real positive or negative zeros is less than its limit, it will differ by an even number. (These relationships are known as Descartes' Rule of Signs.)

As an example, consider the third-degree polynomial function

$$f(x) = x^3 - 3x^2 - 6x + 8 .$$

It can have no more than three real zeros. It has at most two positive real zeros (observe the sign changes from the first to second and third to fourth terms) and at most one negative real zero (obtained from $f(-x) = -x^3 - 3x^2 + 6x + 8$).

Polynomial functions are usually evaluated most compactly using nested multiplication. (This is sometimes referred to as Horner's method.) As an illustration, the function from the previous example can be rewritten as

$$f(x) = [(x - 3)x - 6]x + 8 .$$

This representation is more easily programmed and more efficiently executed than the original form, especially since SOLVE fills the stack with the value of x.

Example: During the winter of '78, Arctic explorer Jean-Claude Coulerre, isolated at his frozen camp in the far north, began scanning the southern horizon in anticipation of the sun's reappearance. Coulerre knew that the sun would not be visible to him until early March, when it reached a declination of 5°18′S. On what day and time in March was the chilly explorer's vigil rewarded?

The time in March when the sun reached 5°18′S declination can be computed by solving the following equation for t:

$$D = a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

where D is the declination in degrees, t is the time in days from the beginning of the month, and

$$a_4 = \ 4.2725 \times 10^{-8}$$
$$a_3 = -1.9931 \times 10^{-5}$$
$$a_2 = \ 1.0229 \times 10^{-3}$$
$$a_1 = \ 3.7680 \times 10^{-1}$$
$$a_0 = -8.1806 \ .$$

This equation is valid for $1 \leqslant t < 32$, representing March, 1978.

First convert 5°18′S to decimal degrees (press 5.18 [CHS] [g] [→H]), obtaining −5.3000 (using [FIX] 4 display mode). (Southern latitudes are expressed as negative numbers for calculation purposes.)

The solution to Coulerre's problems is the value of t satisfying

$$-5.3000 = a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0.$$

Expressed in the form required by [SOLVE], the equation is

$$0 = a_4t^4 + a_3t^3 + a_2t^2 + a_1t - 2.8806$$

where the last, constant term now incorporates the value of the declination.

Using Horner's method, the function to be set equal to zero is

$$f(t) = (((a_4t + a_3)t + a_2)t + a_1)t - 2.8806 \ .$$

To shorten the subroutine, store and recall the constants using the registers corresponding to the exponent of t.

Keystrokes	Display	
[ON] / [−]	**Pr Error**	Clears calculator's memory.*
[←]	**0.0000**	
[g] [P/R]	**000–**	Program mode.

*This step is included here only to ensure that sufficient memory is available for the examples that follow in this handbook.

Keystrokes	Display
[f] [LBL] [A]	001-42,21,11
[RCL] 4	002- 45 4
[×]	003- 20
[RCL] 3	004- 45 3
[+]	005- 40
[×]	006- 20
[RCL] 2	007- 45 2
[+]	008- 40
[×]	009- 20
[RCL] 1	010- 45 1
[+]	011- 40
[×]	012- 20
[RCL] 0	013- 45 0
[+]	014- 40
[g] [RTN]	015- 43 32

In Run mode, key in the five coefficients:

Keystrokes	Display		
[g] [P/R]			Run mode.
4.2725 [EEX] 8 [CHS]	4.2725	-08	
[STO] 4	4.2725	-08	Coefficient of t^4.
1.9931 [CHS] [EEX] 5 [CHS] [STO] 3	-1.9931	-05	Coefficient of t^3.
1.0229 [EEX] 3 [CHS]	1.0229	-03	
[STO] 2	0.0010		Coefficient of t^2.
3.7680 [EEX] 1 [CHS]	3.7680	-01	
[STO] 1	0.3768		Coefficient of t.
2.8806 [CHS] [STO] 0	-2.8806		Constant term.

Because the desired solution should be between 1 and 32, key in these two values for initial estimates. Then use [SOLVE] to find the roots.

Keystrokes	Display	
1 [ENTER]	1.0000	
32	32	Initial estimates.
[f] [SOLVE] [A]	7.5137	Root found.
[R↓]	7.5137	Same previous estimate.

Keystrokes	Display	
[R↓]	0.0000	Function value.
[g] [R↑] [g] [R↑]	7.5137	Restores stack.

The day was March 7th. Convert the fractional portion of the number to decimal hours and then to hours, minutes, and seconds.

Keystrokes	Display	
[f] [FRAC]	0.5137	Fractional portion of day.
24 [×]	12.3293	Decimal hours.
[f] [→H.MS]	12.1945	Hours, minutes, seconds.

Explorer Coulerre should expect to see the sun on March 7th at $12^{h}\ 19^{m}\ 45^{s}$ (Coordinated Universal Time).

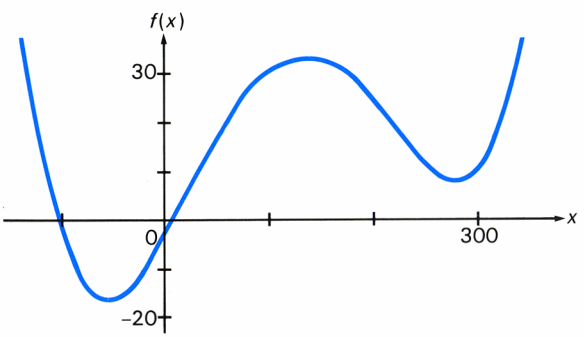
By examining Coulerre's function $f(t)$, you realize that it can have as many as four real roots—three positive and one negative. Try to find additional positive roots by using [SOLVE] with larger positive estimates.

Keystrokes	Display	
1000 [ENTER] 1100	1,100	Two larger, positive estimates.
[f] [SOLVE] [A]	Error 8	No root found.
[←]	278.4497	Last estimate tried.
[R↓]	276.7942	A previous estimate.
[R↓]	7.8948	Nonzero value of function.
[g] [R↑] [g] [R↑]	278.4497	Restores stack to original state.
[f] [SOLVE] [A]	Error 8	Again, no root found.
[←]	278.4398	Approximately same estimate.
[R↓]	278.4497	A previous estimate.
[R↓]	7.8948	Same function value.

You have found a positive local minimum rather than a root. Now try to find the negative root.

Keystrokes	Display	
1000 [CHS] [ENTER]	-1,000.0000	
1100 [CHS]	-1,100	Two larger, negative estimates.
[f] [SOLVE] [A]	-108.9441	Negative root.
[R↓]	-108.9441	Same previous estimate.
[R↓]	1.6000 -08	Function value.

There is no need to search further—you have found all possible roots. The negative root has no meaning since it is outside of the range for which the declination approximation is valid. The graph of the function confirms the results you have found.



Solving a System of Equations

[SOLVE] is designed to find a single variable value that satisfies a single equation. If a problem involves a system of equations with several variables, you may still be able to [SOLVE] to find a solution.

For some systems of equations, expressed as

$$f_1(x_1, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, \ldots, x_n) = 0$$

it is possible through algebraic manipulation to eliminate all but one variable. That is, you can use the equations to derive

expressions for all but one variable in terms of the remaining variable. By using these expressions, you can reduce the problem to using SOLVE to find the root of a single equation. The values of the other variables at the solution can then be calculated using the derived expressions.

This is often useful for solving a complex equation for a complex root. For such a problem, the complex equation can be expressed as two real-valued equations—one for the real component and one for the imaginary component—with two real variables—representing the real and imaginary parts of the complex root.

For example, the complex equation $z + 9 + 8e^{-z} = 0$ has no real roots z, but it has infinitely many complex roots $z = x + iy$. This equation can be expressed as two real equations

$$\begin{aligned} x + 9 + 8e^{-x}\cos y &= 0 \\ y \quad - 8e^{-x}\sin y &= 0\,. \end{aligned}$$

The following manipulations can be used to eliminate y from the equations. Because the sign of y doesn't matter in the equations, assume $y > 0$, so that any solution (x,y) gives another solution $(x,-y)$. Rewrite the second equation as

$$x = \ln(8(\sin y)/y),$$

which requires that $\sin y > 0$, so that $2n\pi < y < (2n + 1)\pi$ for integer $n = 0, 1, \ldots$.

From the first equation

$$\begin{aligned} y &= \cos^{-1}(-e^x(x + 9)/8) + 2n\pi \\ &= (2n + 1)\pi - \cos^{-1}(e^x(x + 9)/8) \end{aligned}$$

for $n = 0, 1, \ldots$ Substitute this expression into the second equation,

$$x + \ln\left(\frac{(2n + 1)\pi - \cos^{-1}(e^x(x + 9)/8)}{\sqrt{64 - (e^x(x + 9))^2}}\right) = 0.$$

You can then use [SOLVE] to find the root x of this equation (for any given value of n, the number of the root). Knowing x, you can calculate the corresponding value of y.

A final consideration for this example is to choose the initial estimates that would be appropriate. Because the argument of the inverse cosine must be between -1 and 1, x must be more negative than about -0.1059 (found by trial and error or by using [SOLVE]). The initial guesses might be near but more negative than this value, -0.11 and -0.2 for example.

(The complex equation used in this example is solved using an iterative procedure in the example on page 81. Another method for solving a system of nonlinear equations is described on page 122.)

Finding Local Extremes of a Function

Using the Derivative

The traditional way to find local maximums and minimums of a function's graph uses the *derivative* of the function. The derivative is a function that describes the slope of the graph. Values of x at which the derivative is zero represent potential local extremes of the function. (Although less common for well-behaved functions, values of x where the derivative is infinite or undefined are also possible extremes.) If you can express the derivative of a function in closed form, you can use [SOLVE] to find where the derivative is zero—showing where the function may be maximum or minimum.

Example: For the design of a vertical broadcasting tower, radio engineer Ann Tenor wants to find the angle from the tower at which the relative field intensity is most negative. The relative intensity created by the tower is given by

$$E = \frac{\cos(2\pi h \cos\theta) - \cos(2\pi h)}{[1 - \cos(2\pi h)]\sin\theta}$$

where E is the relative field intensity, h is the antenna height in wavelengths, and θ is the angle from vertical in radians. The height is 0.6 wavelengths for her design.

The desired angle is one at which the derivative of the intensity with respect to θ is zero.

To save program memory space and execution time, store the following constants in registers and recall them as needed:

$r_0 = 2\pi h$ and is stored in register R_0,

$r_1 = \cos(2\pi h)$ and is stored in register R_1,

$r_2 = 1/[1 - \cos(2\pi h)]$ and is stored in register R_2.

The derivative of the intensity E with respect to the angle θ is given by

$$\frac{dE}{d\theta} = r_2\left[r_0\sin(r_0\cos\theta) - \frac{\cos(r_0\cos\theta) - r_1}{\sin\theta\tan\theta}\right].$$

Key in a subroutine to calculate the derivative.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] 0	001-42,21, 0	
[COS]	002- 24	
[RCL] 0	003- 45 0	
[×]	004- 20	
[COS]	005- 24	
[RCL] 1	006- 45 1	
[-]	007- 30	
[x≷y]	008- 34	
[SIN]	009- 23	
[÷]	010- 10	
[x≷y]	011- 34	
[TAN]	012- 25	
[÷]	013- 10	
[CHS]	014- 16	
[x≷y]	015- 34	
[COS]	016- 24	
[RCL] 0	017- 45 0	

Keystrokes	Display	
[×]	018-	20
[SIN]	019-	23
[RCL] 0	020-	45 0
[×]	021-	20
[+]	022-	40
[RCL] 2	023-	45 2
[×]	024-	20
[g] [RTN]	025-	43 32

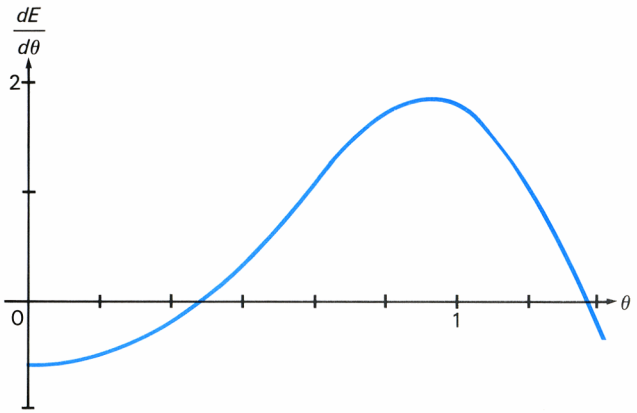
In Radians mode, calculate and store the three constants.

Keystrokes	Display	
[g] [P/R]		Run mode.
[g] [RAD]		Specifies Radians mode.
2 [g] [π] [×]	6.2832	
.6 [×] [STO] 0	3.7699	Constant r_0.
[COS] [STO] 1	-0.8090	Constant r_1.
[CHS] 1 [+]	1.8090	
[1/x] [STO] 2	0.5528	Constant r_2.

The relative field intensity is maximum at an angle of 90° (perpendicular to the tower). To find the minimum, use angles closer to zero as initial estimates, such as the radian equivalents of 10° and 60°.

Keystrokes	Display		
10 [f] [→RAD]	0.1745		
60 [f] [→RAD]	1.0472		Initial estimates.
[f] [SOLVE] 0	0.4899		Angle giving zero slope.
[R↓] [R↓]	-5.5279	-10	Slope at specified angle.
[g] [R↑] [g] [R↑]	0.4899		Restores the stack.
[g] [→DEG]	28.0680		Angle in degrees.

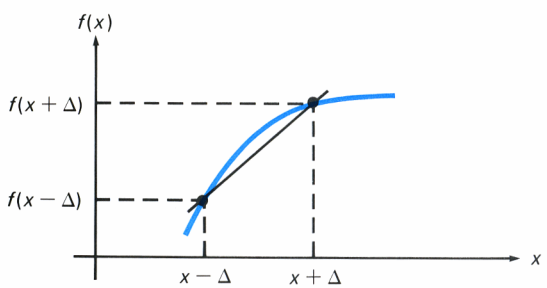
The relative field intensity is most negative at an angle of 28.0680° from vertical.



Using an Approximate Slope

The derivative of a function can also be approximated numerically. If you sample a function at two points relatively close to x (namely $x + \Delta$ and $x - \Delta$), you can use the slope of the secant as an approximation to the slope at x:

$$s = \frac{f(x + \Delta) - f(x - \Delta)}{2\Delta}$$



The accuracy of this approximation depends upon the increment Δ and the nature of the function. Smaller values of Δ give better approximations to the derivative, but excessively small values can cause round-off inaccuracy. A value of x at which the slope is zero is potentially a local extreme of the function.

Example: Solve the previous example without using the equation for the derivative $dE/d\theta$.

Find the angle at which the derivative (determined numerically) of the intensity E is zero.

In Program mode, key in two subroutines: one to estimate the derivative of the intensity and one to evaluate the intensity function E. In the following subroutine, the slope is calculated between $\theta + 0.001$ and $\theta - 0.001$ radians (a range equivalent to approximately 0.1°).

Keystrokes	Display	
[g] [P/R]	000–	Program Mode.
[f] [LBL] [A]	001–42,21,11	
[EEX]	002– 26	
[CHS]	003– 16	
3	004– 3	Evaluates E at $\theta + 0.001$.
[+]	005– 40	
[ENTER]	006– 36	
[GSB] [B]	007– 32 12	
[x≷y]	008– 34	
[EEX]	009– 26	
[CHS]	010– 16	
3	011– 3	Evaluates E at $\theta - 0.001$.
[−]	012– 30	
[ENTER]	013– 36	
[GSB] [B]	014– 32 12	
[−]	015– 30	
2	016– 2	
[EEX]	017– 26	
[CHS]	018– 16	
3	019– 3	

Keystrokes	Display	
[÷]	020- 10	
[g] [RTN]	021- 43 32	
[f] [LBL] [B]	022-42,21,12	Subroutine for $E(\theta)$.
[COS]	023- 24	
[RCL] 0	024- 45 0	
[×]	025- 20	
[COS]	026- 24	
[RCL] 1	027- 45 1	
[-]	028- 30	
[x≷y]	029- 34	
[SIN]	030- 23	
[÷]	031- 10	
[RCL] 2	032- 45 2	
[×]	033- 20	
[g] [RTN]	034- 43 32	

In the previous example, the calculator was set to Radians mode and the three constants were stored in registers R_0, R_1, and R_2. Key in the same initial estimates as before and execute [SOLVE].

Keystrokes	Display	
[g] [P/R]		Run mode.
10 [f] [→RAD]	0.1745	
60 [f] [→RAD]	1.0472	Initial estimates.
[f] [SOLVE] [A]	0.4899	Angle given zero slope.
[R↓] [R↓]	0.0000	Slope at specified angle.
[g] [R↑] [g] [R↑]	0.4899	Restores the stack.
[ENTER] [ENTER] [f] [B]	-0.2043	Uses function subroutine to calculate minimum intensity.
[x≷y]	0.4899	Recalls θ value.
[g] [→DEG]	28.0679	Angle in degrees.

This numerical approximation of the derivative indicates a minimum field intensity of -0.2043 at an angle of 28.0679°. (This angle differs from the previous solution by 0.0001°.)

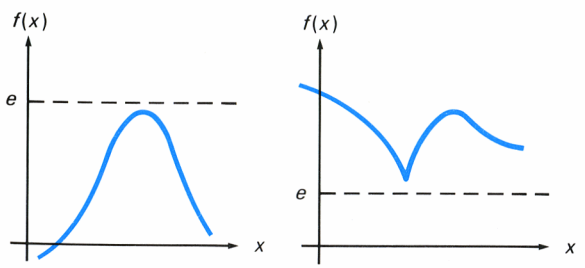
Using Repeated Estimation

A third technique is useful when it isn't practical to calculate the derivative. It is a slower method because it requires the repeated use of the [SOLVE] key. On the other hand, you don't have to find a good value for Δ of the previous method. To find a local extreme of the function $f(x)$, define a new function

$$g(x) = f(x) - e$$

where e is a number slightly beyond the estimated extreme value of $f(x)$. If e is properly chosen, $g(x)$ will approach zero near the extreme of $f(x)$ but will not equal zero. Use [SOLVE] to analyze $g(x)$ near the extreme. The desired result is **Error 8**.

- If **Error 8** is displayed, the number in the X-register is an x value near the extreme. The number in the Z-register tells roughly how far e is from the extreme value of $f(x)$. Revise e to bring it closer (but not equal) to the extreme value. Then use [SOLVE] to examine the revised $g(x)$ near the x value previously found. Repeat this procedure until successive x values do not differ significantly.
- If a root of $g(x)$ is found, either the number e is not beyond the extreme value of $f(x)$ or else [SOLVE] has found a different region where $f(x)$ equals e. Revise e so that it is close to—but beyond—the extreme value of $f(x)$ and try [SOLVE] again. It may also be possible to modify $g(x)$ in order to eliminate the distant root.



Example: Solve the previous example without calculating the derivative of the relative field intensity E.

The subroutine to calculate E and the required constants have been entered in the previous example.

In Program mode, key in a subroutine that subtracts an estimated extreme number from the field intensity E. The extreme number should be stored in a register so that it can be manually changed as needed.

Keystrokes	Display	
[g] [P/R]	000-	Program mode.
[f] [LBL] 1	001-42,21, 1	Begins with label.
[GSB] [B]	002- 32 12	Calculates E.
[RCL] 9	003- 45 9	
[-]	004- 30	Subtracts extreme estimate.
[g] [RTN]	005- 43 32	

In Run mode, estimate the minimum intensity value by manually sampling the function.

Keystrokes	Display	
[g] [P/R]		Run mode.
10 [f] [→RAD]	0.1745	Samples the function at 10°, 30°, 50°, ...
[ENTER] [f] [B]	-0.1029	
30 [f] [→RAD]	0.5236	
[ENTER] [f] [B]	-0.2028	
50 [f] [→RAD]	0.8727	
[ENTER] [f] [B]	0.0405	

Based on these samples, try using an extreme estimate of -0.25 and initial [SOLVE] estimates (in radians) near 10° and 30°.

Keystrokes	Display	
.25 [CHS] [STO] 9	-0.2500	Stores extreme estimate.
.2 [ENTER]	0.2000	
.6	0.6	Initial estimates.
[f] [SOLVE] 1	Error 8	No root found.
[←] [STO] 4	0.4849	Stores θ estimate.
[R↓] [STO] 5	0.4698	Stores previous θ estimate.
[R↓]	0.0457	Distance from extreme.
.9 [×]	0.0411	Revises extreme estimate
[STO] [+] 9	0.0411	by 90 percent of the distance.
[RCL] 4	0.4849	Recalls θ estimate.
[ENTER] [ENTER] [f] [B]	-0.2043	Calculates intensity E.
[←]	0.0000	Recalls other θ estimate,
[RCL] 5	0.4698	keeping first estimate in Y-register.
[f] [SOLVE] 1	Error 8	No root found.
[←]	0.4898	θ estimate.
[x≷y]	0.4893	Previous θ estimate.
[x≷y]	0.4898	Recalls θ estimate.
[ENTER] [ENTER] [f] [B]	-0.2043	Calculates intensity E.
[x≷y]	0.4898	Recalls θ value.
[g] [→DEG]	28.0660	Angle in degrees.
[g] [DEG]	28.0660	Restores Degrees mode.

The second interation produces two θ estimates that differ in the fourth decimal place. The field intensities E for the two iterations are equal to four decimal places. Stopping at this point, a minimum field intensity of -0.2043 is indicated at an angle of 28.0660°. (This angle differs from the previous solutions by about 0.002°.)

Applications

The following applications illustrate how you can use [SOLVE] to simplify a calculation that would normally be difficult—finding an interest rate that can't be calculated directly. Other applications that use the [SOLVE] function are given in sections 3 and 4.

Annuities and Compound Amounts

This program solves a variety of financial problems involving money, time, and interest. For these problems, you normally know the values of three or four of the following variables and need to find the value of another:

n The *number* of compounding periods. (For example, a 30 year loan with monthly payments has $n = 12 \times 30 = 360$.)

i The *interest rate* per compounding period expressed as a percent. (To calculate *i*, divide the annual percentage rate by the number of compounding periods in a year. That is, 12% annual interest compounded monthly equals 1% periodic interest.)

PV The *present value* of a series of future cash flows or the initial cash flow.

PMT The periodic *payment* amount.

FV The *future value*. That is, the final cash flow (balloon payment or remaining balance) or the compounded value of a series of prior cash flows.

Possible Problems Involving Annuities and Compound Amounts

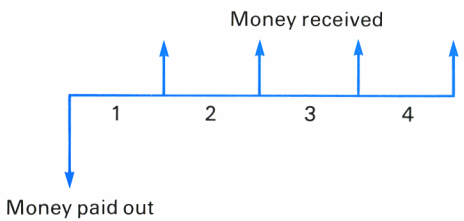
Allowable Combination of Variables	Typical Applications		Initial Procedure
	For Payments at End of Period	For Payments at Beginning of Period	
n, *i*, *PV*, *PMT* (Enter any three and calculate the fourth.)	Direct reduction loan. Discounted note. Mortgage.	Lease. Annuity due.	Use [f] CLEAR [REG] or set *FV* to zero.
n, *i*, *PV*, *PMT*, *FV* (Enter any four and calculate the fifth.)	Direct reduction loan with balloon payment. Discounted note.	Lease with residual value. Annuity due.	None.
n, *i*, *PMT*, *FV* (Enter any three and calculate the fourth.)	Sinking fund.	Periodic savings. Insurance.	Use [f] CLEAR [REG] or set *PV* to zero.
n, *i*, *PV*, *FV* (Enter any three and calculate the fourth.)	Compound growth. Savings.		Use [f] CLEAR [REG] or set *PMT* to zero.

The program accommodates payments that are made at the beginning or end of compounding periods. Payments made at the end of compounding periods (ordinary annuity) are common in direct reduction loans and mortgages. Payments made at the

beginning of compounding periods (annuity due) are common in leasing. For payments at the end of periods, clear flag 0. For payments at the beginning of periods, set flag 0. If the problem involves no payments, the status of flag 0 has no effect.

This program uses the convention that money paid out is entered and displayed as a negative number, and that money received is entered and displayed as a positive number.

A financial problem can usually be represented by a cash flow diagram. This is a pictorial representation of the timing and direction of financial transactions. The cash flow diagram has a horizontal time line that is divided into equal increments that correspond to the compounding period—months or years, for example. Vertical arrows represent exchanges of money, following the convention that an upward arrow (positive) represents money received and a downward arrow (negative) represents money paid out. (The examples that follow are illustrated using cash flow diagrams.)



Pressing [f] CLEAR [REG] provides a convenient way to set up the calculator for a new problem. However, it isn't necessary to press [f] CLEAR [REG] between problems. You need to reenter the values of only those variables that change from problem to problem. If a variable isn't applicable for a new problem, simply enter zero as its value. For example, if *PMT* is used in one problem but not used in the next, simply enter zero for the value of *PMT* in the second problem.

The basic equation used for the financial calculations is

$$PV + \frac{PMT\,A}{i/100}[1-(1+i/100)^{-n}] + FV(1+i/100)^{-n} = 0$$

where $i \neq 0$ and

$$A = \begin{cases} 1 & \text{for end-of-period payments} \\ 1+i/100 & \text{for beginning-of-period payments.} \end{cases}$$

The program has the following characteristics:

- [SOLVE] is used to find i. Because this is an iterative function, solving for i takes longer than finding other variables. It is possible to define problems which cannot be solved by this technique. If [SOLVE] can't find a root, **Error 4** is displayed.
- When finding any of the variables listed on the left below, certain conditions result in an **Error 4** display:

n	$PMT = -PV\,i/(100\,A)$
	$(PMT\,A - FV\,i/100)/(PMT\,A + PV\,i/100) \leq 0$
	$i \leq -100$
i	[SOLVE] can't find a root
PV	$i \leq -100$
PMT	$n = 0$
	$i = 0$
	$i \leq -100$
FV	$i \leq -100$

- If a problem has a specified interest rate of 0, the program generates an **Error 0** display (or **Error 4** when solving for PMT).
- Problems with extremely large (greater than 10^6) or extremely small (less than 10^{-6}) values for n and i may give invalid results.
- Interest problems with balloon payments of opposite signs to the periodic payments may have more than one mathematically correct answer (or no answer at all). This program may find one of the answers but has no way of finding or indicating other possibilities.

Keystrokes	**Display**	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	**000–**	

Keystrokes	Display	
[f] [LBL] [A]	001-42,21,11	n routine.
[STO] 1	002- 44 1	Stores n.
[R/S]	003- 31	
[GSB] 1	004- 32 1	Calculates n.
[g] [LSTx]	005- 43 36	
[RCL] [×] 0	006-45,20, 0	
[RCL] 5	007- 45 5	
[x≷y]	008- 34	
[−]	009- 30	Calculates $FV - 100\,PMT\,A/i$.
[g] [LSTx]	010- 43 36	
[RCL] [+] 3	011-45,40, 3	Calculates $PV + 100\,PMT\,A/i$.
[g] [x=0]	012- 43 20	Tests $PMT = -PV\,i/(100\,A)$.
[GTO] 0	013- 22 0	
[÷]	014- 10	
[CHS]	015- 16	
[g] [TEST] 4	016-43,30, 4	Tests $x \leq 0$.
[GTO] 0	017- 22 0	
[g] [LN]	018- 43 12	
[RCL] 6	019- 45 6	
[g] [LN]	020- 43 12	
[÷]	021- 10	
[STO] 1	022- 44 1	
[g] [RTN]	023- 43 32	
[f] [LBL] [B]	024-42,21,12	i routine.
[STO] 2	025- 44 2	Stores i.
[R/S]	026- 31	
[·]	027- 48	
2	028- 2	
[ENTER]	029- 36	
[EEX]	030- 26	

Keystrokes	Display	
[CHS]	031– 16	
3	032– 3	
[g] [CF] 1	033–43, 5, 1	Clears flag 1 for [SOLVE] subroutine.
[f] [SOLVE] 3	034–42,10, 3	
[GTO] 4	035– 22 4	
[GTO] 0	036– 22 0	
[f] [LBL] 4	037–42,21, 4	
[EEX]	038– 26	
2	039– 2	
[×]	040– 20	Calculates *i*.
[STO] 2	041– 44 2	
[g] [RTN]	042– 43 32	
[f] [LBL] [C]	043–42,21,13	*PV* routine.
[STO] 3	044– 44 3	Stores *PV*.
[R/S]	045– 31	
[GSB] 1	046– 32 1	Calculates *PV*.
[GSB] 2	047– 32 2	
[CHS]	048– 16	
[STO] 3	049– 44 3	
[g] [RTN]	050– 43 32	
[f] [LBL] [D]	051–42,21,14	*PMT* routine.
[STO] 4	052– 44 4	Stores *PMT*.
[R/S]	053– 31	
1	054– 1	Calculates *PMT*.
[STO] 4	055– 44 4	
[GSB] 1	056– 32 1	
[RCL] 3	057– 45 3	
[GSB] 2	058– 32 2	
[x≷y]	059– 34	
[÷]	060– 10	
[CHS]	061– 16	
[STO] 4	062– 44 4	
[g] [RTN]	063– 43 32	

Keystrokes	Display	
[f] [LBL] [E]	064-42,21,15	FV routine.
[STO] 5	065- 44 5	Stores FV.
[R/S]	066- 31	
[GSB] 1	067- 32 1	Calculates FV.
[RCL] [+] 3	068-45,40, 3	
[RCL] [÷] 7	069-45,10, 7	
[CHS]	070- 16	
[STO] 5	071- 44 5	
[g] [RTN]	072- 43 32	
[f] [LBL] 1	073-42,21, 1	
[g] [SF] 1	074-43, 4, 1	Sets flag 1 for subroutine 3.
1	075- 1	
[RCL] 2	076- 45 2	
[g] [%]	077- 43 14	Calculates $i/100$.
[f] [LBL] 3	078-42,21, 3	[SOLVE] subroutine.
[STO] 8	079- 44 8	
1	080- 1	
[STO] 0	081- 44 0	
[+]	082- 40	
[g] [TEST] 4	083-43,30, 4	Tests $i \leq 100$.
[GTO] 0	084- 22 0	
[STO] 6	085- 44 6	
[g] [F?] 0	086-43, 6, 0	Tests for end-of-period payments.
[STO] 0	087- 44 0	
[RCL] 1	088- 45 1	
[CHS]	089- 16	
[y^x]	090- 14	Calculates $(1 + i/100)^{-n}$.
[STO] 7	091- 44 7	
1	092- 1	
[$x \rightleftarrows y$]	093- 34	
[-]	094- 30	Calculates $1 - (1 + i/100)^{-n}$.

Keystrokes	Display	
[g] [x=0]	095- 43 20	Tests $i = 0$ or $n = 0$.
[GTO] 0	096- 22 0	
[RCL] [×] 0	097-45,20, 0	
[RCL] 4	098- 45 4	
[RCL] [÷] 8	099-45,10, 8	
[×]	100- 20	
[g] [F?] 1	101-43, 6, 1	Tests flag 1 set.
[g] [RTN]	102- 43 32	
[RCL] [+] 3	103-45,40, 3	[SOLVE] subroutine continues.
[f] [LBL] 2	104-42,21, 2	
[RCL] 5	105- 45 5	
[RCL] [×] 7	106-45,20, 7	Calculates $FV(1 + i/100)^{-n}$.
[+]	107- 40	
[g] [RTN]	108- 43 32	[SOLVE] subroutine ends.

Labels used: A, B, C, D, E, 0, 1, 2, 3, and 4.

Registers used: R_0 (A), R_1 (n), R_2 (i), R_3 (PV), R_4 (PMT), R_5 (FV), R_6, R_7, and R_8.

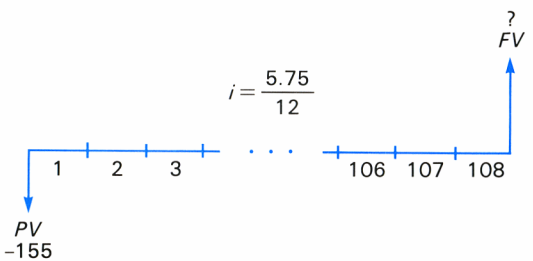
To use the program:

1. Press 8 [f] [DIM] [(i)] to reserve R_0 through R_8.
2. Press [f] [USER] to activate User mode.
3. If necessary, press [f] CLEAR [REG] to clear all of the financial variables. You don't need to clear the registers if you intend to specify all of the values.
4. Set flag 0 according to how payments are to be figured:
   - Press [g] [CF] 0 for payments at the end of the period.
   - Press [g] [SF] 0 for payments at the beginning of the period.
5. Enter the known values of the financial variables:
   - To enter n, key in the value and press [A].
   - To enter i, key in the value and press [B].

- To enter PV, key in the value and press [C].
- To enter PMT, key in the value and press [D].
- To enter FV, key in the value and press [E].

6. Calculate the unknown value:
   - To calculate n, press [A] [R/S].
   - To calculate i, press [B] [R/S].
   - To calculate PV, press [C] [R/S].
   - To calculate PMT, press [D] [R/S].
   - To calculate FV, press [E] [R/S].
7. To solve another problem, repeat steps 3 through 6 as needed. Be sure that any variable not to be used in the problem has a value of zero.

Example: You place \$155 in a savings account paying 5¾% compounded monthly. What sum of money can you withdraw at the end of 9 years?



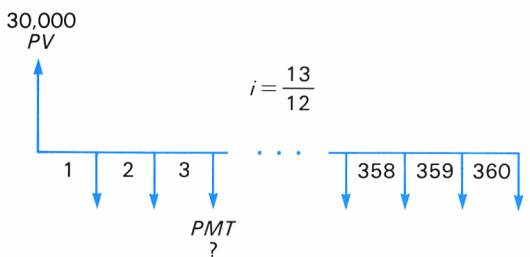
Keystrokes	Display	
[g] [P/R]		Run mode.
[f] CLEAR [REG]		Clears financial variables.
[f] [FIX] 2		
[f] [USER]		Activates User mode.
[g] [CF] 0		Ordinary annuity.
9 [ENTER] 12 [×] [A]	**108.00**	Enters $n = 9 \times 12$.

Keystrokes	Display	
5.75 [ENTER] 12 [÷] [B]	**0.48**	Enters $i = 5.75/12$.
155 [CHS] [C]	**-155.00**	Enters $PV = -155$ (money paid out).
[E] [R/S]	**259.74**	Calculates FV.

If you desire a sum of \$275, what would be the required interest rate?

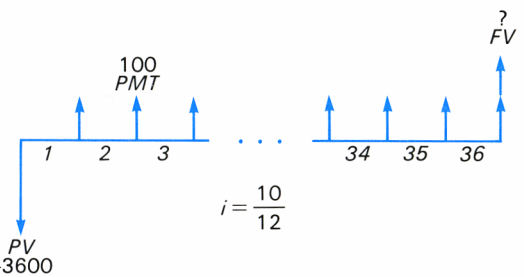
Keystrokes	Display	
275 [E]	**275.00**	Enters $FV = 275$.
[B] [R/S]	**0.53**	Calculates i.
12 [×]	**6.39**	Calculates annual interest rate.

Example: You receive \$30,000 from the bank as a 30-year, 13% mortgage. What monthly payment must you make to the bank to fully amortize the mortgage?



Keystrokes	Display	
[f] CLEAR [REG]		Clears financial variables.
30 [ENTER] 12 [×] [A]	**360.00**	Enters $n = 30 \times 12$.
13 [ENTER] 12 [÷] [B]	**1.08**	Enters $i = 13/12$.
30000 [C]	**30,000.00**	Enters $PV = 30{,}000$.
[D] [R/S]	**-331.86**	Calculates PMT (money paid out).

Example: You offer a loan of $3,600 that is to be repaid in 36 monthly payments of $100 with an annual interest rate of 10%. What balloon payment amount, to be paid coincident with the 36th payment, is required to pay off the loan?



Keystrokes	Display	
[f] CLEAR [REG]		Clears financial variables.
36 [A]	**36.00**	Enters $n = 36$.
10 [ENTER] 12 [÷] [B]	**0.83**	Enters $i = 10/12$.
3600 [CHS] [C]	**-3600.00**	Enters $PV = -3600$ (money paid out).
100 [D]	**100.00**	Enters $PMT = 100$ (money received).
[E] [R/S]	**675.27**	Calculates FV.

The final payment is $675.27 + $100.00 = $775.27 because the final payment and balloon payment are due at end of the last period.

Example: You're collecting a $50,000 loan at 14% annual interest over 360 months. Find the remaining balance after the 24th payment and the interest accrued between the 12th and 24th payments.

You can use the program to calculate accumulated interest and the remaining balance for loans. The accumulated interest is equal to the total payments made during that time less the principal reduction during that time. The principal reduction is the difference between the remaining balances at the start and end of the period.

First, calculate the payment on the loan.

Keystrokes	Display	
[f] CLEAR [REG]		Clears financial variables.
360 [A]	**360.00**	Enter $n = 360$.
14 [ENTER] 12 [÷] [B]	**1.17**	Enters $i = 14/12$.
50000 [CHS] [C]	**-50,000.00**	Enters $PV = -50{,}000$.
[D] [R/S]	**592.44**	Calculates PMT.

Now calculate the remaining balance at month 24.

Keystrokes	Display	
24 [A]	**24.00**	Enters $n = 24$.
[E] [R/S]	**49,749.56**	Calculates FV at month 24.

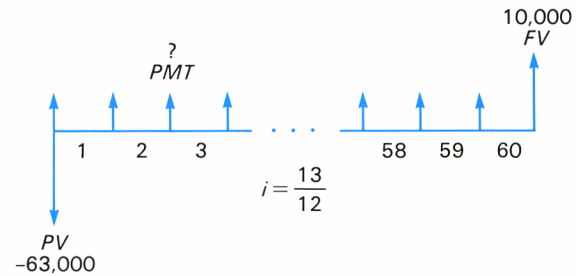
Store this remaining balance, then calculate the remaining balance at month 12 and the principal reduction between payments 12 and 24.

Keystrokes	Display	
[STO] [I]	**49,749.56**	
12 [A]	**12.00**	Enters $n = 12$.
[E] [R/S]	**49,883.48**	Calculates FV at month 12.
[RCL] [I]	**49,749.56**	Recalls FV at month 24.
[-]	**133.92**	Calculates principal reduction.

The accrued interest is the value of 12 payments less the principal reduction.

Keystrokes	Display	
[RCL] 4	**592.44**	Recalls PMT.
12 [×]	**7,109.23**	Calculates value of payments.
[x≷y] [-]	**6,975.31**	Calculates accrued interest.

Example: A leasing firm is considering the purchase of a minicomputer for $63,000 and wants to achieve a 13% annual yield by leasing the computer for a 5-year period. At the end of the lease the firm expects to sell the computer for at least $10,000. What monthly payment should the firm charge in order to achieve a 13% yield? (Because the lease payments are due at the beginning of each month, be sure to set flag 0 to specify beginning-of-period payments.)



Keystrokes	Display	
f CLEAR REG		Clears financial variables.
g SF 0		Specifies beginning-of-period payments.
5 ENTER 12 × A	**60.00**	Enters $n = 5 \times 12$.
13 ENTER 12 ÷ B	**1.08**	Enters $i = 13/12$.
63000 CHS C	**-63,000.00**	Enters $PV = -63{,}000$.
10000 E	**10,000.00**	Enters $FV = 10{,}000$.
D R/S	**1,300.16**	Calculates PMT.

If the price of the computer increases to $70,000, what should the payments be?

Keystrokes	Display	
70000 CHS C	**-70,000.00**	Enters $PV = -70{,}000$.
D R/S	**1,457.73**	Calculates PMT.

If the payments were increased to $1,500, what would the yield be?

Keystrokes	Display	
1500 [D]	**1,500.00**	Enters $PMT = 1500$.
[B] [R/S]	**1.18**	Calculates i (monthly).
12 [×]	**14.12**	Calculates annual yield.
[f] [USER]	**14.12**	Deactivates User mode.

Discounted Cash Flow Analysis

This program performs two kinds of discounted cash flow analysis: net present value (*NPV*) and internal rate of return (*IRR*). It calculates *NPV* or *IRR* for up to 24 groups of cash flows.

The cash flows are stored in the two-column matrix **C**. Matrix **C** has one row for each group of cash flows. In each row of **C**, the first element is the cash flow amount; the second element is the number of consecutive cash flows having that amount (the number of flows in that group.) The first element of **C** must be the amount of the initial investment. The cash flows must occur at equal intervals; if no cash flow occurs for several time periods, enter 0 for the cash flow amount and the number of zero cash flows in that group.

After all the cash flows have been stored in matrix **C**, you can enter an assumed interest rate and calculate the net present value (*NPV*) of the investment. Alternatively, you can calculate the internal rate of return (*IRR*). The *IRR* is the interest rate that makes the present value of a series of cash flows equal to the initial investment. It's the interest rate that makes the *NPV* equal zero. *IRR* is also called the *yield* or *discounted rate of return*.

The fundamental equation for *NPV* is

$$NPV = \begin{cases} \displaystyle\sum_{j=1}^{k} CF_j \left(\frac{1-(1+i/100)^{-n_j}}{i/100} \right) (1+i/100)^{-\sum_{l<j} n_l} & \text{for } i > -100,\ i \neq 0 \\ \displaystyle\sum_{j=1}^{k} CF_j n_j & \text{for } i = 0 \end{cases}$$

where $\sum_{l<1} n_l$ is defined as -1.

The program uses the convention that money received is entered and displayed as a positive number, and that money paid out is entered and displayed as a negative number.

The program has the following characteristics:

- The cash flow sequence (including the initial investment) must contain both a positive flow and a negative flow. That is, there must be at least one sign change.
- Cash flows with multiple sign changes may have more than one solution. This program may find one solution, but it has no way of indicating other possibilities.
- The *IRR* calculation may take several minutes (5 or more) depending of the number of cash flow entries.
- The program displays **Error 4** if it is unable to find a solution for *IRR* or if the yield $i \leq -100\%$ in the *NPV* calculation.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] [A]	001-42,21,11	*NPV* routine.
[EEX]	002- 26	
2	003- 2	
[÷]	004- 10	Calculates *IRR*/100.
[GSB] 2	005- 32 2	
[R/S]	006- 31	
[f] [LBL] [B]	007-42,21,12	*IRR* routine.
1	008- 1	
[ENTER]	009- 36	
[EEX]	010- 26	
[CHS]	011- 16	
3	012- 3	
[f] [SOLVE] 2	013-42,10, 2	
[GTO] 1	014- 22 1	
[GTO] 0	015- 22 0	Branch for no *IRR* solution.
[f] [LBL] 1	016-42,21, 1	
[EEX]	017- 26	

Keystrokes	Display	
2	018– 2	
[×]	019– 20	
[R/S]	020– 31	
[f] [LBL] 2	021–42,21, 2	Calculates *NPV*.
[g] [CF] 0	022–43, 5, 0	
[STO] 2	023– 44 2	
1	024– 1	
[STO] 4	025– 44 4	
[+]	026– 40	Calculates $1 + IRR/100$.
[g] [TEST] 4	027–43,30, 4	Tests $IRR \leq -100$.
[GTO] 0	028– 22 0	Branch for $IRR \leq -100$.
[STO] 3	029– 44 3	
0	030– 0	
[STO] 5	031– 44 5	
[f] [MATRIX] 1	032–42,16, 1	
[f] [LBL] 3	033–42,21, 3	
[g] [F?] 0	034–43, 6, 0	Tests if all flows used.
[GTO] 7	035– 22 7	Branch for all flows used.
[GSB] 6	036– 32 6	
[RCL] 2	037– 45 2	
[g] [x=0]	038– 43 20	Tests $IRR = 0$.
[GTO] 4	039– 22 4	Branch for $IRR = 0$.
1	040– 1	
[+]	041– 40	
[GSB] 6	042– 32 6	
[CHS]	043– 16	
[y^x]	044– 14	
[STO] 4	045– 44 4	
1	046– 1	
[$x \leftrightarrows y$]	047– 34	
[−]	048– 30	
[RCL] [÷] 2	049–45,10, 2	
[RCL] [×] 3	050–45,20, 3	
[GTO] 5	051– 22 5	
[f] [LBL] 4	052–42,21, 4	
[$x \leftrightarrows y$]	053– 34	
[GSB] 6	054– 32 6	
[f] [LBL] 5	055–42,21, 5	

Keystrokes	Display	
[×]	056- 20	
[STO] [+] 5	057-44,40, 5	
[RCL] 4	058- 45 4	
[STO] [×] 3	059-44,20, 3	
[GTO] 3	060- 22 3	
[f] [LBL] 6	061-42,21, 6	Recalls cash flow element.
[f] [USER] [RCL] [C]	062u 45 13	
[f] [USER]		
[g] [RTN]	063- 43 32	
[g] [SF] 0	064-43, 4, 0	Sets flag 0 if last element.
[g] [RTN]	065- 43 32	
[f] [LBL] 7	066-42,21, 7	
[RCL] 5	067- 45 5	Recalls *NPV*.
[g] [RTN]	068- 43 32	

Labels used: A, B, and 0 through 7.

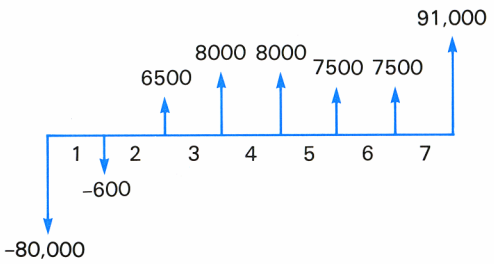
Registers used: R_0 through R_5.

Matrix used: **C**.

To use the discounted cash flow analysis program:

1. Press 5 [f] [DIM] [(i)] to allocate registers R_0 through R_5.
2. Press [f] [USER] to activate User mode (unless it's already active).
3. Key in the number of cash flow groups, then press [ENTER] 2 [f] [DIM] [C] to dimension matrix **C**.
4. Press [f] [MATRIX] 1 to set the row and column numbers to 1.
5. For each cash flow group:
   a. Key in the amount and press [STO] [C], then
   b. Key in the number of occurrences and press [STO] [C].
6. Calculate the desired parameter:
   - To calculate *IRR*, press [B].
   - To calculate *NPV*, enter periodic interest rate *i* in percent and press [A]. Repeat for as many interest rates as needed.
7. Repeat steps 3 through 6 for other sets of cash flows.

Example: An investor pays $80,000 for a duplex that he intends to sell after 7 years. He must spend some money the first year for repairs. At the end of the seventh year the duplex is sold for $91,000. Will he achieve a desired 9% after-tax yield with the following after-tax cash flows?



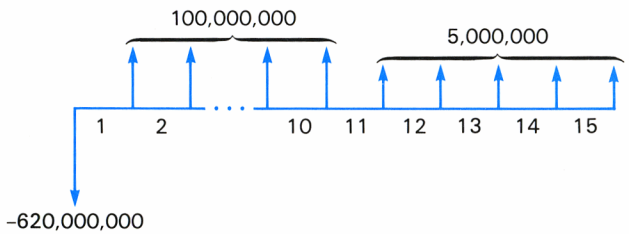
Keystrokes	Display	
[g] [P/R]		Run mode.
[f] [FIX] 2		
5 [f] [DIM] [(i)]	5.00	Reserve registers R_0 through R_5.
6 [ENTER] 2	2	
[f] [DIM] [C]	2.00	
[f] [MATRIX] 1	2.00	
[f] [USER]	2.00	
80000 [CHS] [STO] [C]	-80,000.00	Initial investment.
1 [STO] [C]	1.00	
600 [CHS] [STO] [C]	-600.00	
1 [STO] [C]	1.00	
6500 [STO] [C]	6,500.00	
1 [STO] [C]	1.00	
8000 [STO] [C]	8,000.00	
2 [STO] [C]	2.00	
7500 [STO] [C]	7,500.00	
2 [STO] [C]	2.00	
91000 [STO] [C]	91,000.00	
1 [STO] [C]	1.00	
9	9	Enters assumed yield.
[A]	-4,108.06	*NPV*.

Since the *NPV* is negative, the investment does not achieve the desired 9% yield. Calculate the *IRR*.

Keystrokes	Display	
[B]	**8.04**	*IRR* (after about 8 minutes).

The *IRR* is less than the desired 9% yield.

Example: An investment of $620,000,000 is expected to have an annual income stream for the next 15 years as shown in the diagram.



What is the expected rate of return?

Keystrokes	Display	
3 [ENTER] 2	**2**	
[f] [DIM] [C]	**2.00**	
[f] [MATRIX] 1	**2.00**	
620000000 [CHS]	**-620,000,000**	
[STO] [C]	**-620,000,000.0**	
1 [STO] [C]	**1.00**	
100000000 [STO] [C]	**100,000,000.0**	
10 [STO] [C]	**10.00**	
5000000 [STO] [C]	**5,000,000.00**	
5 [STO] [C]	**5.00**	
[B]	**10.06**	*IRR*.
[f] [FIX] 4	**10.0649**	
[f] [USER]	**10.0649**	Deactivates User mode.

Section 2

Working With $\int_y^x$

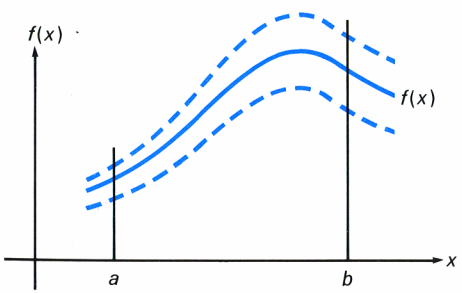
The HP-15C gives you the ability to perform numerical integration using $\int_y^x$. This section shows you how to use $\int_y^x$ effectively and describes techniques that enable you to handle difficult integrals.

Numerical Integration Using $\int_y^x$

A calculator using numerical integration can almost never calculate an integral precisely. But the $\int_y^x$ function asks you in a convenient way to specify how much error is tolerable. It asks you to set the display format according to how many figures are accurate in the integrand $f(x)$. In effect, you specify the width of a ribbon drawn around the graph of $f(x)$. The integral estimated by $\int_y^x$ corresponds to the area under some unspecified graph lying entirely within the ribbon. Of course, this estimate could vary by as much as the area of the ribbon, so $\int_y^x$ estimates this area too. If I is the desired integral, then

$$I = \begin{pmatrix} \text{area under a graph} \\ \text{drawn in the ribbon} \end{pmatrix} \pm \begin{pmatrix} \tfrac{1}{2} \text{ area of} \\ \text{the ribbon} \end{pmatrix}$$

The HP-15C places the first area estimate in the X-register and the second—the uncertainty—in the Y-register.



For example, $f(x)$ might represent a physical effect whose magnitude can be determined only to within $\pm$ 0.005. Then the value calculated as $f(x)$ has an uncertainty of 0.005. A display setting of FIX 2 tells the calculator that decimal digits beyond the second can't matter. The calculator need not waste time estimating the integral with unwarranted precision. Instead, the calculator can more quickly give you a fair idea of the range of values within which the integral must lie.

The HP-15C doesn't prevent you from declaring that $f(x)$ is far more accurate than it really is. You can specify the display setting after a careful error analysis, or you can just offer a guess. You may leave the display set to SCI 4 or FIX 4 without much further thought. You will get an estimate of the integral and its uncertainty, enabling you to interpret the result more intelligently than if you got the answer with no idea of its accuracy or inaccuracy.

The ∫f algorithm uses a Romberg method for accumulating the value of the integral. Several refinements make it more effective.

Instead of using uniformly spaced samples, which can induce a kind of resonance or aliasing that produces misleading results when the integrand is periodic, ∫f uses samples that are spaced nonuniformly. Their spacing can be demonstrated by substituting, say,

$$x = \frac{3}{2}u - \frac{1}{2}u^3$$

into

$$I = \int_{-1}^{1} f(x)dx = \int_{-1}^{1} f\left(\frac{3}{2}u - \frac{1}{2}u^3\right)\frac{3}{2}\ (1-u^2)\ du$$

and sampling u uniformly. Besides suppressing resonance, the substitution has two more benefits. First, no sample need be drawn from either end of the interval of integration (except when the interval is so narrow that no other possibilities are available). As a result, an integral like

$$\int_0^3 \frac{\sin x}{x}\,dx$$

won't be interrupted by division by zero at an endpoint. Second, ∫ can integrate functions that behave like $\sqrt{|x-a|}$, whose slope is infinite at an endpoint. Such functions are encountered when calculating the area enclosed by a smooth, closed curve.

Another refinement is that ∫ uses extended precision, 13 significant digits, to accumulate the internal sums. This allows thousands of samples to be accumulated, if necessary, without losing to roundoff any more information than is lost within your function subroutine.

Accuracy of the Function to be Integrated

The accuracy of an integral calculated using ∫ depends on the accuracy of the function calculated by your subroutine. This accuracy, which you specify using the display format, depends primarily on three considerations:

- The accuracy of empirical constants in the function.
- The degree to which the function may accurately describe a physical situation.
- The extent of round-off error in the internal calculations of the calculator.

Functions Related to Physical Situations

Functions like $\cos(4\theta - \sin\theta)$ are *pure mathematical functions*. In this context, this means that the functions do not contain any empirical constants, and neither the variables nor the limits of integration represent actual physical quantities. For such functions, you can specify as many digits as you want in the display format (up to nine) to achieve the desired degree of accuracy in the integral.* All you need to consider is the trade-off between the accuracy and calculation time.

---

* Provided that $f(x)$ is still calculated accurately, despite round-off error, to the number of digits shown in the display.

There are additional considerations, however, when you're integrating functions relating to an actual physical situation. Basically, with such functions you should ask yourself *whether the accuracy you would like in the integral is justified by the accuracy in the function.* For example, if the function contains empirical constants that are specified to only, say, three significant digits, it might not make sense to specify more than three digits in the display format.

Another important consideration—and one which is more subtle and therefore more easily overlooked—is that nearly every function relating to a physical situation *is inherently inaccurate to a certain degree*, because it is only a mathematical *model* of an actual process or event. A mathematical model is itself an *approximation* that ignores the effects of known or unknown factors which are insignificant to the degree that the results are still useful.

An example of a mathematical model is the *normal distribution function*

$$\int_{-\infty}^{t} \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}\,dx$$

which has been found to be useful in deriving information concerning physical measurements on living organisms, product dimensions, average temperatures, etc. Such mathematical descriptions typically are either derived from theoretical considerations or inferred from experimental data. To be practially useful, they are constructed with certain assumptions, such as ignoring the effects of relatively insignificant factors. For example, the accuracy of results obtained using the normal distribution function as a model of the distribution of certain quantities depends on the size of the population being studied. And the accuracy of results obtained from the equation $s = s_0 - \frac{1}{2}gt^2$, which gives the height of a falling body, ignores the variation with altitude of g, the acceleration of gravity.

Thus, mathematical descriptions of the physical world can provide results of only limited accuracy. If you calculated an integral with an apparent accuracy beyond that with which the model describes

the actual behavior of the process or event, you would not be justified in using the calculated value to the full apparent accuracy.

Round-Off Error in Internal Calculations

With any computational device—including the HP-15C—calculated results must be "rounded off" to a finite number of digits (10 digits in the HP-15C). Because of this *round-off error*, calculated results—especially results of evaluating a function that contains several mathematical operations—may not be accurate to all 10 digits that can be displayed. Note that round-off error affects the evaluation of *any* mathematical expression, not just the evaluation of a function to be integrated using ∫. (Refer to the appendix for additional information.)

If $f(x)$ is a function relating to a physical situation, its inaccuracy due to round-off typically is insignificant compared to the inaccuracy due to empirical constants, etc. If $f(x)$ is what we have called a pure mathematical function, its accuracy is limited only by round-off error. Generally, it would require a complicated analysis to determine precisely how many digits of a calculated function might be affected by round-off. In practice, its effects are typically (and adequately) determined through experience rather than analysis.

In certain situations, round-off error can cause peculiar results, particularly if you should compare the results of calculating integrals that are equivalent mathematically but differ by a transformation of variables. However, you are unlikely to encounter such situations in typical applications.

Shortening Calculation Time

The time required for ∫ to calculate an integral depends on how soon a certain density of sample points is achieved in the region where the function is interesting. The calculation of the integral of any function will be prolonged if the interval of integration includes mostly regions where the function is not interesting. Fortunately, if you must calculate such an integral, you can modify the problem so that the calculation time is reduced. Two such techniques are subdividing the interval of integration and transformation of variables.

Subdividing the Interval of Integration

In regions where the slope of $f(x)$ is varying appreciably, a high density of sample points is necessary to provide an approximation that changes insignificantly from one iteration to the next. However, in regions where the slope of the function stays nearly constant, a high density of sample points is not necessary. This is because evaluating the function at additional sample points would not yield much new information about the function, so it would not dramatically affect the disparity between successive approximations. Consequently, in such regions an approximation of comparable accuracy could be achieved with substantially fewer sample points: so much of the time spent evaluating the function in these regions is wasted. When integrating such functions, you can save time by using the following procedure:

1. Divide the interval of integration into subintervals over which the function is interesting and subintervals over which the function is uninteresting.
2. Over the subintervals where the function is interesting, calculate the integral in the display format corresponding to the accuracy you would like overall.
3. Over the subintervals where the function either is not interesting or contributes negligibly to the integral, calculate the integral with less accuracy, that is, in a display format specifying fewer digits.
4. To get the integral over the entire interval of integration, add together the approximations and their uncertainties from the integrals calculated over each subinterval. You can do this easily using the [Σ+] key.

Before subdividing the integration, check whether the calculator underflows when evaluating the function around the upper (or lower) limit of integration.* Since there is no reason to evaluate the function at values of x for which the calculator underflows, in some cases the upper limit of integration can be reduced, saving considerable calculation time.

---

* When the calculation of any quantity would result in a number less than 10^{-99}, the result is replaced by zero. This condition is known as underflow.

Remember that once you have keyed in the subroutine that evaluates $f(x)$, you can calculate $f(x)$ for any value of x by keying that value into the X-register and pressing [ENTER] [ENTER] [ENTER] [GSB] followed by the label of the subroutine.

If the calculator underflows at the upper limit of integration, try smaller numbers until you get closer to the point where the calculator no longer underflows.

For example, consider the approximation of

$$\int_0^{\infty} xe^{-x}dx \ .$$

Key in a subroutine that evaluates the function $f(x) = xe^{-x}$.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	Clears program memory.
[f] [LBL] 1	001-42,21, 1	
[CHS]	002- 16	
[e^x]	003- 12	
[×]	004- 20	
[g] [RTN]	005- 43 32	

Set the calculator to Run mode and set the display format to [SCI] 3. They try several values of x to find where the calculator underflows for your function.

Keystrokes	Display		
[g] [P/R]			Run mode.
[f] [SCI] 3			Sets format to [SCI] 3.
[EEX] 3	1	03	Keys 1000 into X-register.
[ENTER] [ENTER] [ENTER]	1.000	03	Fills the stack with x.
[GSB] 1	0.000	00	Calculator underflows at $x = 1000$.
300 [ENTER]	3.000	02	Tries a smaller value of x.
[ENTER] [ENTER]	3.000	02	
[GSB] 1	0.000	00	Calculator still underflows.
200 [ENTER]	2.000	02	Try a smaller value of x.

Keystrokes	Display		
ENTER ENTER	2.000	02	
GSB 1	2.768	-85	Calculator doesn't underflow at $x = 200$; try a number between 200 and 250.
225 ENTER	2.250	02	
ENTER ENTER	2.250	02	
GSB 1	4.324	-96	Calculator is close to underflow.

At this point, you can use SOLVE to pinpoint the smallest value of x at which the calculator underflows.

Keystrokes	Display		
R↓	2.250	02	Roll down stack until the last value tried is in the X- and Y-registers.
f SOLVE 1	2.280	02	The minimum value of x at which the calculator underflows is about 228.

You've now determined that you need integrate only from 0 to 228. Since the integrand is interesting only for values of x less than 10, divide the interval of integration there. The problem has now become:

$$\int_0^{\infty} xe^{-x}dx \approx \int_0^{228} xe^{-x}dx = \int_0^{10} xe^{-x}dx + \int_{10}^{228} xe^{-x}dx.$$

Keystrokes	Display		
7 f DIM (i)	7.000	00	Allocates statistical storage registers.
f CLEAR Σ+	0.000	00	Clears statistical storage registers.
0 ENTER	0.000	00	Keys in lower limit of integration over first subinterval.
10	10		Keys in upper limit of integration over first subinterval.

Keystrokes	Display		
f ∫ 1	9.995	-01	Integral over (0, 10) calculated in SCI 3.
Σ+	1.000	00	Sum approximation and its uncertainty in registers R_3 and R_5.
x≷y	1.841	-04	Uncertainty of approximation.
R↓ R↓	1.000	01	Roll down stack until upper limit of first integral appears in X-register.
228	228		Keys upper limit of second integral into X-register. Upper limit of first integral is lifted into Y-register, becoming lower limit of second integral.
f SCI 0	2.	02	Specifies SCI 0 display format for a quick calculation over (10, 228). If the uncertainty of the approximation turns out not to be accurate enough, you can repeat the approximation in a display format specifying more digits.
f ∫ 1	5.	-04	Integral over (10, 228) calculated in SCI 0.
f SCI 3	5.328	-04	Changes display format back to SCI 3.
x≷y	7.568	-05	Checks uncertainty of approximation. Since it is less than the uncertainty of the approximation over the first subinterval, SCI 0 yielded an approximation of sufficient accuracy.

Keystrokes	Display	
x⇄y	5.328	-04 Returns approximation and its uncertainty to the X- and Y-registers, respectively, before summing them in statistical storage registers.
Σ+	2.000	00 Sums approximation and its uncertainty.
RCL Σ+	1.000	00 Integral over total interval (0, 228) (recalled from R_3).
x⇄y	2.598	-04 Uncertainty of integral (from R_5).

Transformation of Variables

In many problems where the function changes very slowly over most of a very wide interval of integration, a suitable transformation of variables may decrease the time required to calculate the integral.

For example, consider again the integral

$$\int_0^{\infty} xe^{-x}dx .$$

Let $e^{-x} = u^3$.

Then $x = -3 \ln u$

and $dx = -3\frac{du}{u}$.

Substituting,

$$\int_0^{\infty} xe^{-x}dx = \int_{e^{-0}}^{e^{-\infty}} (-3 \ln u)(u^3)\left(-3\frac{du}{u}\right)$$

$$= \int_1^0 9u^2 \ln u \, du.$$

Key in a subroutine that evaluates the function $f(u) = 9u^2 \ln u$.

Keystrokes	Display	
g P/R	000-	Program mode.
f LBL 3	001-42,21, 3	
g LN	002- 43 12	
x≷y	003- 34	
g x²	004- 43 11	
×	005- 20	
9	006- 9	
×	007- 20	
g RTN	008- 43 32	

Key in the limits of integration, then press f ∫ 3 to calculate the integral.

Keystrokes	Display	
g P/R		Run mode.
1 ENTER	1.000 00	Keys in lower limit of integration.
0	0	Keys in upper limit of integration.
f ∫ 3	1.000 00	Approximation to equivalent integral.
x≷y	3.020 -04	Uncertainty of approximation.

The approximation agrees with the value calculated in the previous problem for the same integral.

Evaluating Difficult Integrals

Certain conditions can prolong the time required to evaluate an integral or can cause inaccurate results. As discussed in the *HP-15C Owner's Handbook*, these conditions are related to the nature of the integrand over the interval of integration.

One class of integrals that are difficult to calculate is improper integrals. An improper integral is one that involves ∞ in at least one of the following ways:

- One or both limits of integration are $\pm\infty$, such as

$$\int_{-\infty}^{\infty} e^{-u^2} du = \sqrt{\pi} .$$

- The integrand tends to $\pm\infty$ someplace in the range of integration, such as

$$\int_0^1 \ln(u)\, du = 1.$$

- The integrand oscillates infinitely rapidly somewhere in the range of integration, such as

$$\int_0^1 \cos(\ln u)\, du = \tfrac{1}{2}.$$

Equally troublesome are nearly improper integrals, which are characterized by

- The integrand or its first derivative changes wildly within a relatively narrow subinterval of the range of integration, or oscillates frequently across that range.

The HP-15C attempts to deal with certain of the second type of improper integral by usually not sampling the integrand at the limits of integration.

Because improper and nearly improper integrals are not uncommon in practice, you should recognize them and take measures to evaluate them accurately. The following examples illustrate techniques that are helpful.

Consider the integrand

$$f(x) = \frac{\sqrt{-2 \ln \cos(x^2)}}{x^2}.$$

This function loses its accuracy when x becomes small. This is caused by rounding $\cos(x^2)$ to 1, which drops information about how small x is. But by using $u = \cos(x^2)$, you can evaluate the integrand as

$$f(x) = \begin{cases} 1 & \text{if } u = 1 \\ \dfrac{\sqrt{-2 \ln u}}{\cos^{-1} u} & \text{if } u \neq 1. \end{cases}$$

Although the branch for $u = 1$ adds four steps to your subroutine, integration near $x = 0$ becomes more accurate.

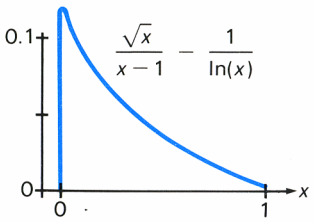
As a second example, consider the integral

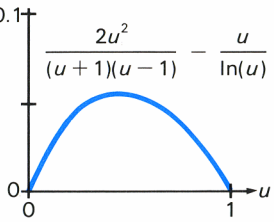
$$\int_0^1 \left(\frac{\sqrt{x}}{x-1} - \frac{1}{\ln x} \right) dx.$$

The derivative of the integrand approaches ∞ as x approaches 0, as shown in the illustration below. By substituting $x = u^2$, the function becomes more well behaved, as shown in the second illustration. This integral is easily evaluated:

$$\int_0^1 \left(\frac{2u^2}{(u+1)(u-1)} - \frac{u}{\ln u} \right) du.$$

Don't replace $(u + 1)(u - 1)$ by $(u^2 - 1)$ because as u approaches 1, the second expression loses to roundoff half of its significant digits and introduces to the integrand's graph a spike near $u = 1$.





As another example, consider a function whose graph has a long tail that stretches out many, many times as far as the main "body" (where the graph is interesting)—a function like

$$f(x) = e^{-x^2} \quad \text{or} \quad g(x) = \frac{1}{x^2 + 10^{-10}}.$$

Thin tails, like that of $f(x)$, can be truncated without greatly degrading the accuracy or speed of integration. But $g(x)$ has too wide a tail to ignore when calculating

$$\int_{-t}^{t} g(x)\,dx$$

if t is large.

For such functions, a substitution like $x = a + b \tan u$ works well, where a lies within the graph's main "body" and b is roughly its width. Doing this for $f(x)$ from above with $a = 0$ and $b = 1$ gives

$$\int_0^t f(x)\,dx = \int_0^{\tan^{-1}t} e^{-\tan^2 u}(1 + \tan^2 u)\,du,$$

which is calculated readily even with t as large as 10^{10}. Using the same substitution with $g(x)$, values near $a = 0$ and $b = 10^{-5}$ provide good results.

This example involves subdividing the interval of integration. Although a function may have features that look extreme over the entire interval of integration, over portions of that interval the function may look more well-behaved. Subdividing the interval of integration works best when combined with appropriate substitutions. Consider the integral

$$\begin{aligned}
\int_0^\infty dx/(1+x^{64}) &= \int_0^1 dx/(1+x^{64}) + \int_1^\infty dx/(1+x^{64}) \\
&= \int_0^1 dx/(1+x^{64}) + \int_0^1 u^{62}du/(u^{64}+1) \\
&= \int_0^1 (1+x^{62})dx/(1+x^{64}) \\
&= 1 + \int_0^1 (x^{62}-x^{64})dx/(1+x^{64}) \\
&= 1 + \tfrac{1}{8}\int_0^1 (1-v^{1/4})v^{55/8}\,dv/(1+v^8).
\end{aligned}$$

These steps use the substitutions $x = 1/u$ and $x = v^{1/8}$ and some algebraic manipulation. Although the original integral is improper, the last integral is easily handled by [ƒ]. In fact, by separating the constant term from the integral, you obtain (using [SCI] 8) an answer with 13 significant digits:

$$1.000401708155 \pm 1.2 \times 10^{-12}.$$

A final example drawn from real life involves the electrostatic field about an ellipsoidal probe with principal semiaxes a, b, and c:

$$V = \int_0^\infty \frac{dx}{(a^2+x)\sqrt{(a^2+x)(b^2+x)(c^2+x)}}$$

for $a = 100$, $b = 2$, and $c = 1$.*

Transform this improper integral to a proper one by substituting $x = (a^2 - c^2)/(1 - u^2) - a^2$:

$$V = p\int_r^1 \sqrt{(1-u^2)/(u^2+q)}\, du$$

where

$$p = 2/((a^2-c^2)\sqrt{a^2-b^2}) = 2.00060018 \times 10^{-6}$$

$$q = (b^2-c^2)/(a^2-b^2) = 3.001200480 \times 10^{-3}$$

$$r = c/a = 0.01\,.$$

However, this integral is nearly improper because q and r are both so nearly zero. But by using an integral in closed form that sufficiently resembles the troublesome part of V, the difficulty can be avoided. Try

$$W = p\int_r^1 du/\sqrt{u^2+q} = p\,\ln(u + \sqrt{u^2+q}\,)\Big|_r^1$$

$$= p\,\ln((1+\sqrt{1+q})/(r+\sqrt{r^2+q}\,))$$

$$= 8.40181880708 \times 10^{-6}.$$

Then

$$V = W + p\int_r^1 (\sqrt{(1-u^2)/(u^2+q)} - 1/\sqrt{u^2+q}\,)\,du$$

$$= p\int_r^1 \left(\frac{W/p}{1-r} - \frac{u^2}{(1+\sqrt{1-u^2}\,)\sqrt{u^2+q}}\right) du.$$

*From Stratton, J.A., *Electromagnetic Theory*, McGraw-Hill, New York, 1941, pp. 201-217.

The HP-15C readily handles this integral. Don't worry about $\sqrt{1-u^2}$ *as* u approaches 1 because the figures lost to roundoff aren't needed.

Application

The following program calculates the values of four special functions for any argument x:

$$P(x)=\frac{1}{2\pi}\int_{-\infty}^{x} e^{-t^2/2}dt$$ (normal distribution function)

$$Q(x)=1-P(x)=\frac{1}{2\pi}\int_{x}^{\infty} e^{-t^2/2}dt$$ (complementary normal distribution function)

$$\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_{0}^{x} e^{-t^2}dt$$ (error function)

$$\mathrm{erfc}(x)=1-\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_{x}^{\infty} e^{-t^2}dt$$ (complementary error function)

The program calculates these functions using the transformation $u=e^{-t^2}$ whenever $|x|>1.6$.

The function value is returned in the X-register, and the uncertainty of the *integral* is returned in the Y-register. (The uncertainty of the *function* value is approximately the same order of magnitude as the number in the Y-register.) The original argument is available in register R_0.

The program has the following characteristics:

- The display format specifies the accuracy of the integrand in the same way as it does for ∫ itself. However, if you specify an unnecessarily large number of display digits, the calculation will be prolonged.
- Small function values, such as Q(20), P(-20), and erfc(10), are accurately computed as quickly as moderate values.

Keystrokes	Display	
g P/R		Program mode.
f CLEAR PRGM	000–	
f LBL A	001–42,21,11	Program for $P(x)$.
STO 2	002– 44 2	Stores x in R_2.
CHS	003– 16	Calculates $-x$.
GTO 2	004– 22 2	Branches to calculate $P(x) = Q(-x)$.
f LBL B	005–42,21,12	Program for $Q(x)$.
STO 2	006– 44 2	Stores x in R_2.
f LBL 2	007–42,21, 2	
2	008– 2	
$\sqrt{x}$	009– 11	
÷	010– 10	
GSB C	011– 32 13	Calculates $\text{erfc}(x/\sqrt{2})$.
2	012– 2	
÷	013– 10	Calculates $Q(x) = \frac{1}{2}\,\text{erfc}(x/\sqrt{2})$.
RCL 2	014– 45 2	
STO 0	015– 44 0	Stores x in R_0.
R↓	016– 33	
g RTN	017– 43 32	Returns function value.
f LBL C	018–42,21,13	Program for $\text{erfc}(x)$.
1	019– 1	
GSB 4	020– 32 4	
g F? 1	021–43, 6, 1	Tests flag 1 set.
GTO 5	022– 22 5	Branches for flag 1 set.
1	023– 1	
−	024– 30	Calculates $\text{erf}(x) - 1$ for flag 1 clear.
f LBL 5	025–42,21, 5	
CHS	026– 16	Calculates $\text{erfc}(x)$.
g RTN	027– 43 32	Returns function value.
f LBL E	028–42,21,15	Program for $\text{erf}(x)$.
0	029– 0	
f LBL 4	030–42,21, 4	Subroutine for $\text{erf}(x)$ or $\text{erfc}(x)$.
g CF 1	031–43, 5, 1	Clears flag 1.

Keystrokes	Display	
STO 1	032– 44 1	Stores 0 for erf(x), 1 for erfc(x).
$x \gtrless y$	033– 34	
STO 0	034– 44 0	
g ABS	035– 43 16	Calculates $\|x\|$.
1	036– 1	
.	037– 48	
6	038– 6	
g TEST 8	039–43,30, 8	Tests $\|x\| > 1.6$.
GTO 6	040– 22 6	Branch for $\|x\| > 1.6$.
0	041– 0	
RCL 0	042– 45 0	Recalls x.
f ∫ 0	043–42,20, 0	Integrates e^{-t^2} from 0 to x.
2	044– 2	
×	045– 20	
f LBL 3	046–42,21, 3	Subroutine to divide by $\sqrt{\pi}$.
g π	047– 43 26	
√x	048– 11	
÷	049– 10	
g RTN	050– 43 32	
f LBL 6	051–42,21, 6	Subroutine to integrate when $\|x\| > 1.6$.
g SF 1	052–43, 4, 1	Sets flag 1.
0	053– 0	
RCL 0	054– 45 0	
g x^2	055– 43 11	
CHS	056– 16	
e^x	057– 12	Calculates e^{-x^2}.
f ∫ 1	058–42,20, 1	Integrates $(-\ln u)^{-1/2}$ from 0 to e^{-x^2}.
GSB 3	059– 32 3	Divides integral by $\sqrt{\pi}$.
RCL 0	060– 45 0	
ENTER	061– 36	
g ABS	062– 43 16	
÷	063– 10	Calculates sign of x.
×	064– 20	

Keystrokes	Display	
RCL 1	065- 45 1	Recalls 1 for erfc(x), 0 for erf(x).
g LSTx	066- 43 36	
-	067- 30	
+	068- 40	Adjusts integral for sign of x and function.
CHS	069- 16	
g RTN	070- 43 32	
f LBL 0	071-42,21, 0	Subroutine to calculate e^{-t^2}.
g x^2	072- 43 11	
CHS	073- 16	
e^x	074- 12	
g RTN	075- 43 32	
f LBL 1	076-42,21, 1	Subroutine to calculate $(-\ln u)^{-1/2}$.
g x=0	077- 43 20	
g RTN	078- 43 32	
g LN	079- 43 12	
CHS	080- 16	
$\sqrt{x}$	081- 11	
1/x	082- 15	
g RTN	083- 43 32	

Labels used: A, B, C, E, 0, 1, 2, 3, 4, 5, and 6.

Registers used: R_0 (x), R_1, R_2.

Flag used: 1.

To use this program:

1. Enter the argument x into the display.
2. Evaluate the desired function:
   - Press f A to evaluate P(x).
   - Press f B to evaluate Q(x).
   - Press f E to evaluate erf(x).
   - Press f C to evaluate erfc(x).

Example: Calculate Q(20), P(1.234), and erf(0.5) in [SCI] 3 display format.

Keystrokes	Display		
[g] [P/R]			Run mode.
[f] [SCI] 3			Specifies format.
20 [f] [B]	**2.754**	**-89**	Q(20).
1.234 [f] [A]	**8.914**	**-01**	P(1.234).
.5 [f] [E]	**5.205**	**-01**	erf(0.5).

Example: For a Normally distributed random variable X with mean 2.151 and standard deviation 1.085, calculate the probability $Pr[2 < X \leq 3]$.

$$Pr[2 < X \leq 3] = Pr\left[\frac{2 - 2.151}{1.085} < \frac{X - \mu}{\sigma} \leq \frac{3 - 2.151}{1.085}\right]$$

$$= \mathrm{P}\left(\frac{3 - 2.151}{1.085}\right) - \mathrm{P}\left(\frac{2 - 2.151}{1.085}\right)$$

Keystrokes	Display		
2 [ENTER]	**2.000**	**00**	
2.151 [-]	**-1.510**	**-01**	
1.085 [÷]	**-1.392**	**-01**	
[f] [A]	**4.447**	**-01**	Calculates $Pr[X \leq 2]$.
[STO] 3	**4.447**	**-01**	Stores value.
3 [ENTER]	**3.000**	**00**	
2.151 [-]	**8.490**	**-01**	
1.085 [÷]	**7.825**	**-01**	
[f] [A]	**7.830**	**-01**	Calculates $Pr[X \leq 3]$.
[RCL] 3	**4.447**	**-01**	Recalls $Pr[X \leq 2]$.
[-]	**3.384**	**-01**	Calculates $Pr[2 < X \leq 3]$.
[f] [FIX] 4	**0.3384**		

Section 3

Calculating in Complex Mode

Physically important problems involving real data are often solved by performing relatively simple calculations using complex numbers. This section gives important insights into complex computation and shows several examples of solving problems involving complex numbers.

Using Complex Mode

Complex mode in the HP-15C enables you to evaluate complex-valued expressions simply. Generally, in Complex mode a mathematical expression is entered in the same manner as in the normal "real" mode. For example, consider a program that evaluates the polynomial $P(x) = a_n x^n + \ldots + a_1 x + a_0$ for the value x in the X-register. By activating Complex mode, this same program can evaluate $P(z)$, where z is complex. Similarly, other expressions, such as the Gamma function $\Gamma(x)$ in the next example, can be evaluated for complex arguments in Complex mode.

Example: Write a program that evaluates the continued-fraction approximation

$$\ln(\Gamma(x)) = (x - \tfrac{1}{2})\ln x - x + a_0 + \cfrac{a_1}{x + \cfrac{a_2}{x + \cfrac{a_3}{x + \ldots}}}$$

for the first six values of a:

$$\begin{aligned} a_0 &= \tfrac{1}{2}\ln(2\pi) \\ a_1 &= 1/12 \\ a_2 &= 1/30 \\ a_3 &= 53/210 \\ a_4 &= 195/371 \\ a_5 &= 1.011523068 \\ a_6 &= 1.517473649. \end{aligned}$$

Because this approximation is valid for both real arguments and complex arguments with $\mathrm{Re}(z) > 0$, this program approximates $\ln(\Gamma(z))$ in Complex mode (for sufficiently large $|z|$). When $|z| > 4$ (and $\mathrm{Re}(z) > 0$), the approximation has about 9 or 10 accurate digits.

Enter the following program.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] [A]	001-42,21,11	
6	002- 6	
[STO] [I]	003- 44 25	Stores counter in Index register.
[x≷y]	004- 34	
[ENTER]	005- 36	
[ENTER]	006- 36	
[ENTER]	007- 36	Fills stack with z.
[RCL] 6	008- 45 6	Recalls a_6.
[f] [LBL] 1	009-42,21, 1	Loop for continued fraction.
[+]	010- 40	
[RCL] [(i)]	011- 45 24	Recalls a_i.
[x≷y]	012- 34	Restores z.
[÷]	013- 10	
[f] [DSE] [I]	014-42, 5,25	Decrements counter.
[GTO] 1	015- 22 1	
[RCL] 0	016- 45 0	Recalls a_0.
[+]	017- 40	
[x≷y]	018- 34	Restores z.
[-]	019- 30	
[g] [LSTx]	020- 43 36	Recalls z.
[g] [LN]	021- 43 12	Calculates $\ln(z)$.
[g] [LSTx]	022- 43 36	Recalls z.
[.]	023- 48	
5	024- 5	
[-]	025- 30	Calculates $z - \frac{1}{2}$.

Keystrokes	Display	
[×]	026- 20	
[+]	027- 40	Calculates $\ln(\Gamma(z))$.
[g] [RTN]	028- 43 32	

Store the constants in registers R_0 through R_6 in order according to their subscripts.

Keystrokes	Display	
[g] [P/R]		Run mode.
2 [g] [π] [×]	6.2832	
[g] [LN] 2 [÷]	0.9189	
[STO] 0	0.9189	Stores a_0.
12 [1/x] [STO] 1	0.0833	Stores a_1.
30 [1/x] [STO] 2	0.0333	Stores a_2.
53 [ENTER] 210 [÷]	0.2524	
[STO] 3	0.2524	Stores a_3.
195 [ENTER] 371 [÷]	0.5256	
[STO] 4	0.5256	Stores a_4.
1.011523068 [STO] 5	1.0115	Stores a_5.
1.517473649 [STO] 6	1.5175	Stores a_6.

Use this program to calculate $\ln(\Gamma(4.2))$, then compare it with $\ln(3.2!)$ calculated with the [x!] function. Also calculate $\ln(\Gamma(1 + 5i))$.

Keystrokes	Display	
4.2 [f] [A]	2.0486	Calculates $\ln(\Gamma(4.2))$.
[f] [FIX] 9	2.048555637	Displays 10 digits.
3.2 [f] [x!]	7.756689536	Calculates $(3.2)! = \Gamma(3.2 + 1)$.
[g] [LN]	2.048555637	Calculates $\ln(3.2!)$.
1 [ENTER]	1.000000000	Enters real part of $1 + 5i$.
5 [f] [I]	1.000000000	Forms complex number $1 + 5i$.

Keystrokes	Display	
f A	**-6.130324145**	Real part of $\ln(\Gamma(1+5i))$.
f Re≷Im	**3.815898575**	Imaginary part of $\ln(\Gamma(1+5i))$.
f FIX 4	**3.8159**	

The complex result is calculated with no more effort than that needed to enter the imaginary part of the argument z. (The result $\ln(\Gamma(1+5i))$ has 10 correct digits in each component.)

Trigonometric Modes

Although the trignometric mode annunciator remains lit in Complex mode, complex functions are *always* computed using *radian* measure. The annunciator indicates the mode (Degrees, Radians, or Grads) for only the two complex conversions: →P and →R.

If you want to evaluate $re^{i\theta}$ where θ is in degrees, e^x can't be used directly because θ must be in radians. If you attempt to convert from degrees to radians, there is a slight loss of accuracy, especially at values like 180° for which the radian measure π can't be represented exactly with 10 digits.

However, in Complex mode the →R function computes $re^{i\theta}$ accurately for θ in *any* measure (indicated by the annunciator). Simply enter r and θ into the complex X-registers in the form $r + i\theta$, then execute →R to calculate the complex value

$$re^{i\theta} = r\cos\theta + ir\sin\theta.$$

(The program listed under Calculating the nth Roots of a Complex Number at the end of this section uses this function.)

Definitions of Math Functions

The lists that follow define the operation of the HP-15C in Complex mode. In these definitions, a complex number is denoted by $z = x + iy$ (rectangular form) or $z = re^{i\theta}$ (polar form). Also $|z| = \sqrt{x^2 + y^2}$.

Arithmetic Operations

$$(a + ib) \pm (c + id) = (a \pm c) + i(b \pm d)$$

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

$$z^2 = z \times z$$

$$1/z = x/|z|^2 - iy/|z|^2$$

$$z_1 \div z_2 = z_1 \times 1/z_2$$

Single-Valued Functions

$$e^z = e^x(\cos y + i \sin y)$$

$$10^z = e^{z \ln 10}$$

$$\sin z = \frac{1}{2i}(e^{iz} - e^{-iz})$$

$$\cos z = \tfrac{1}{2}(e^{iz} + e^{-iz})$$

$$\tan z = \sin z / \cos z$$

$$\sinh z = \tfrac{1}{2}(e^z - e^{-z})$$

$$\cosh z = \tfrac{1}{2}(e^z + e^{-z})$$

$$\tanh z = \sinh z / \cosh z$$

Multivalued Functions

In general, the inverse of a function $f(z)$—denoted by $f^{-1}(z)$—has more than one value for any argument z. For example, $\cos^{-1}(z)$ has infinitely many values for each argument. But the HP-15C calculates the single *principal value*, which lies in the part of the range defined as the principal branch of $f^{-1}(z)$. In the discussion that follows, the single-valued inverse function (restricted to the principal branch) is denoted by uppercase letters—such as $\mathrm{COS}^{-1}(z)$—to distinguish it from the multivalued inverse—$\cos^{-1}(z)$.

For example, consider the nth roots of a complex number z. Write z in polar form as $z = re^{i(\theta + 2k\pi)}$ for $-\pi < \theta \leq \pi$ and $k = 0, \pm 1, \pm 2, \ldots$. Then if n is a positive integer,

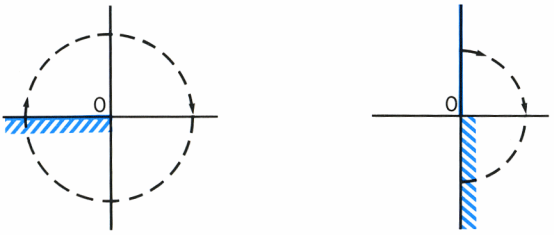
$$z^{1/n} = r^{1/n} e^{i(\theta/n + 2k\pi/n)} = r^{1/n} e^{i\theta/n} e^{i2k\pi/n}.$$

Only $k = 0, 1, \ldots, n - 1$ are necessary since $e^{i2k\pi/n}$ repeats its values in cycles of n. The equation defines the nth roots of z, and $r^{1/n} e^{i\theta/n}$ with $-\pi < \theta \leq \pi$ is the principal branch of $z^{1/n}$. (A program listed on page 78 computes the nth roots of z.)

The illustrations that follow show the principal branches of the inverse relations. The left-hand graph in each figure represents the cut domain of the inverse function; the right-hand graph shows the range of the principal branch.

For some inverse relations, the definitions of the principal branches are not universally agreed upon. The principal branches used by the HP-15C were carefully chosen. First, they are analytic in the regions where the arguments of the real-valued inverse functions are defined. That is, the branch cut occurs where its corresponding real-valued inverse function is undefined. Second, most of the important symmetries are preserved. For example, $\text{SIN}^{-1}(-z) = -\text{SIN}^{-1}(z)$ for all z.

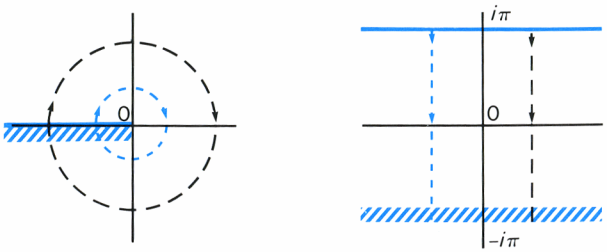
z $\qquad$ $w = \sqrt{z}$



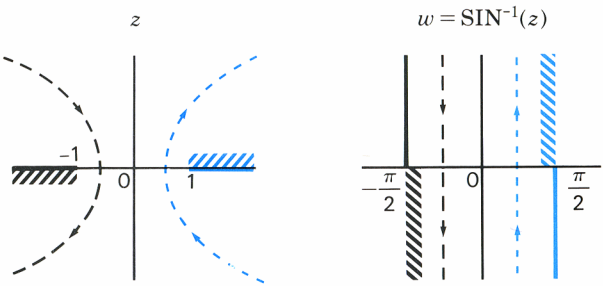
$\sqrt{z} = \sqrt{r}\, e^{i\theta/2}$ $\qquad$ for $-\pi < \theta \leq \pi$

---

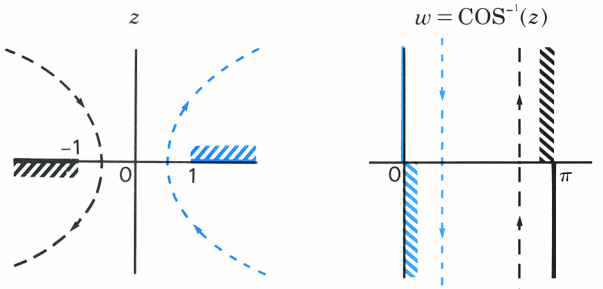
z $\qquad$ $w = \text{LN}(z)$



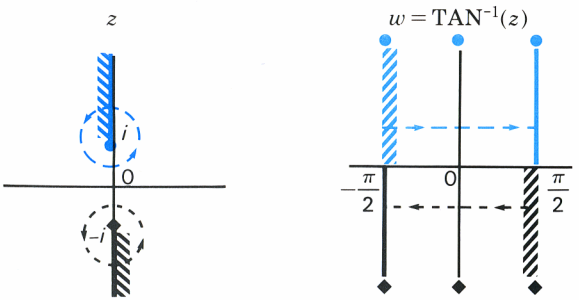
$\text{LN}(z) = \ln r + i\theta$ $\qquad$ for $-\pi < \theta \leq \pi$



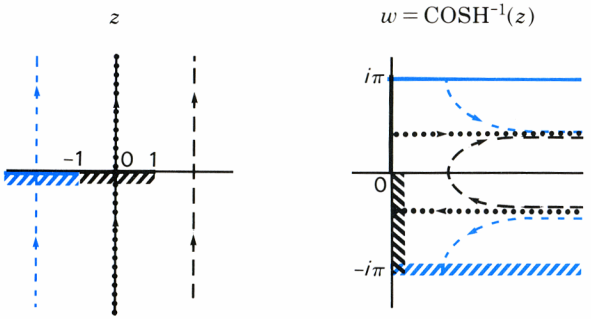
$$\sin^{-1}(z) = -i \ln[iz + (1 - z^2)^{1/2}]$$



$$\cos^{-1}(z) = -i \ln[z + (z^2 - 1)^{1/2}]$$



$$\tan^{-1}(z) = \frac{i}{2} \ln \frac{i + z}{i - z}$$



$$\cosh^{-1}(z) = \ln[z + (z^2 - 1)^{1/2}]$$

The principal branches in the last four graphs above are obtained from the equations shown, but don't necessarily use the principal branches of $\ln(z)$ and $\sqrt{z}$.

The remaining inverse functions may be determined from the illustrations above and the following equations:

$$\text{LOG}(z) = \text{LN}(z)/\text{LN}(10)$$

$$\text{SINH}^{-1}(z) = -i\,\text{SIN}^{-1}(iz)$$

$$\text{TANH}^{-1}(z) = -i\,\text{TAN}^{-1}(iz)$$

$$w^z = e^{z\,\text{LN}(w)}.$$

To determine *all* values of an inverse relation, use the following expressions to derive these values from the principal value calculated by the HP-15C. In these expressions, $k = 0, \pm1, \pm2, \ldots$.

$$z^{1/2} = \pm\sqrt{z}$$

$$\ln(z) = \text{LN}(z) + i2k\pi$$

$$\sin^{-1}(z) = (-1)^k\,\text{SIN}^{-1}(z) + k\pi$$

$$\cos^{-1}(z) = \pm\text{COS}^{-1}(z) + 2k\pi$$

$$\tan^{-1}(z) = \text{TAN}^{-1}(z) + k\pi$$

$$\sinh^{-1}(z) = (-1)^k\,\text{SINH}^{-1}(z) + ik\pi$$

$$\cosh^{-1}(z) = \pm\text{COSH}^{-1}(z) + i2k\pi$$

$$\tanh^{-1}(z) = \text{TANH}^{-1}(z) + ik\pi$$

$$w^z = w^z e^{i2\pi kz}$$

Using [SOLVE] and [∫] in Complex Mode

The [SOLVE] and [∫] functions use algorithms that sample your function at values along the real axis. In Complex mode, the [SOLVE] and [∫] functions operate with only the real stack, even though your function subroutine may have complex computations in it.

For example, [SOLVE] will not search for the roots of a complex function, but rather will sample the function on the real axis and search for a zero of the function's real part. Similarly, [∫] computes the integral of the function's real part along an interval on the real axis. These operations are useful in various applications, such as calculating contour integrals and complex potentials. (Refer to Applications at the end of this section.)

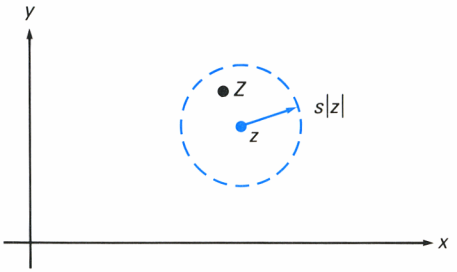
Accuracy in Complex Mode

Because complex numbers have both real components and imaginary components, the accuracy of complex calculations takes on another dimension compared to real-valued calculations.

When dealing with *real* numbers, an approximation X is close to x if the relative difference $E(X,x) = |(X - x)/x|$ is small. This relates directly to the number of correct significant digits of the approximation X. That is, if $E(X,x) < 5 \times 10^{-n}$, then there are at least n significant digits. For *complex* numbers, define $E(Z,z) = |(Z - z)/z|$. This does *not* relate directly to the number of correct digits in each *component* of Z, however.

For example, if $E(X,x)$ and $E(Y,y)$ are both small, then for $z = x + iy$, $E(Z,z)$ must also be small. That is, if $E(X,x) < s$ and $E(Y,y) < s$, then $E(Z,z) < s$. But consider $z = 10^{10} + i$ and $Z = 10^{10}$. The imaginary component of Z is far from accurate, and yet $E(Z,z) < 10^{-10}$. Even though the imaginary components of z and Z are completely different, in a relative sense z and Z are extremely close.

There is a simple, geometric interpretation of the complex relative error. Any approximation Z of z satisfies $E(Z,z) < s$ (where s is a positive real number) if and only if Z lies inside the circle of radius $s|z|$ centered at z in the complex plane.



To require approximations with accurate *components* is to demand more than keeping relative errors small. For example, consider this problem in which the calculations are done with four significant digits. It illustrates the limitations imposed on a complex calculation by finite accuracy.

$$z_1 = Z_1 = 37.1 + 37.3i$$
$$z_2 = Z_2 = 37.5 + 37.3i$$

and

$$\begin{aligned} &Z_1 \times Z_2 \\ &= (37.10 \times 37.50 - 37.30 \times 37.30) + i(37.10 \times 37.30 + 37.30 \times 37.50) \\ &= (1391. - 1391.) + i(1384. + 1399.) \\ &= 0 + i(2783.) \end{aligned}$$

The true value $z_1 z_2 = -0.04 + 2782.58i$. Even though Z_1 and Z_2 have no error, the real part of their four-digit product has no correct significant decimals, although the relative error of the complex product is less than 2×10^{-4}.

The example illustrates that complex multiplication doesn't propogate its errors componentwise. But even if complex multiplication produced accurate components, the rounding errors of a chain computation could quickly produce inaccurate components. On the other hand, the relative error, which corresponds to the precision of the calculation, grows only slowly.

For example, using four-digit accuracy as before

$$z_1 = (1 + 1/300) + i$$
$$Z_1 = 1.003 + i$$
$$z_2 = Z_2 = 1 + i$$

then

$$\begin{aligned} Z_1 \times Z_2 &= (1.003 + i) \times (1 + i) \\ &= 0.003 + 2.003i \\ &= 3.000 \times 10^{-3} + 2.003i \end{aligned}$$

The *correct* four-digit value is $3.333 \times 10^{-3} + 2.003i$. In this example, Z_1 and Z_2 are accurate in each component and the arithmetic is exact. But the product is inaccurate—that is, the real component has only one significant digit. One rounding error causes an inaccurate component, although the complex relative error of the product remains small.

For the HP-15C the results of any complex operation are designed to be accurate in the sense that the complex relative error $E(Z,z)$ is kept small. Generally, $E(Z,z) < 6 \times 10^{-10}$.

As shown earlier, this small relative error doesn't guarantee 10 accurate digits in each component. Because the error is relative to the size $|z|$, and because this is not greatly different from the size of the largest component of z, the smaller component can have fewer accurate digits. There is a quick way for you to see which digits are generally accurate. Express each component using the largest exponent. In this form, approximately the first 10 digits of each component are accurate. For example, if

$$Z = 1.234567890 \times 10^{-10} + i(2.222222222 \times 10^{-3}),$$

then think of Z as

$$0.0000001234567890 \times 10^{-3} + i(2.222222222 \times 10^{-3}).$$

The accurate digits are

$$0.000000123 \times 10^{-3} + i(2.222222222 \times 10^{-3}).$$

Applications

The capability of the HP-15C to work with complex numbers enables you to solve problems that extend beyond the realm of real-valued numbers. On the following pages are several programs that illustrate the usefulness of complex calculations—and the HP-15C.

Storing and Recalling Complex Numbers Using a Matrix

This program uses the stack and matrix **C** to store and recall complex numbers. It has the following characteristics:

- If you specify an index greater than the matrix's dimensions, the calculator displays **Error 3** and the stack is ready for another try.
- If the calculator isn't in Complex mode, the program activates Complex mode and the imaginary part of the number is set to zero.
- When you store a complex number, the index is lost, the stack drops, and the T-register is duplicated in the Z-register.
- The "Store" program uses the [D] key, which is above the [STO] key. The "Recall" program uses the [E] key, which is above the [RCL] key.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000–	
[f] [LBL] [D]	001–42,21,14	"Store" program.
[f] [MATRIX] 1	002–42,16, 1	Sets $R_0 = R_1 = 1$.
[STO] 0	003– 44 0	$R_0 = k$.
[R↓]	004– 33	
0	005– 0	Enters 0 in real (and imaginary) X-registers.
[+]	006– 40	Drops stack and has $a + ib$ in X-register.
[f] [USER] [STO] [C] [f] [USER]	007u 44 13	Stores a and increments indices (User mode).
[f] [Re≷Im]	008– 42 30	

Keystrokes	Display	
STO C	009– 44 13	Stores b (*no* User mode here).
f Re≷Im	010– 42 30	Restores $a + ib$ in X-registers.
g RTN	011– 43 32	
f LBL E	012–42,21,15	"Recall" program.
STO 0	013– 44 0	$R_0 = k$.
g CLx	014– 43 35	Disables stack.
2	015– 2	
STO 1	016– 44 1	Sets $R_1 = 2$.
R↓	017– 33	
0	018– 0	
+	019– 40	Sets stack for another try if **Error 3** occurs next.
RCL C	020– 45 13	Recalls b (imaginary part).
f Re≷Im	021– 42 30	
f DSE 1	022–42, 5, 1	Decrements to $R_1 = 1$.
g CLx	023– 43 35	Disables stack and clears real X-register.
RCL C	024– 45 13	Recalls a (real part).
g RTN	025– 43 32	

Example: Store $2 + 3i$ and $7 + 4i$ in elements 1 and 2 using the previous program. Then recall and add them. Dimension matrix **C** to 5×2 so that it can store up to 5 complex numbers.

After entering the preceding program:

Keystrokes	Display	
g P/R		Run mode.
5 ENTER 2	2	Specifies 5 rows and 2 columns.
f DIM C	2.0000	Dimensions matrix **C**.
2 ENTER 3 f I	2.0000	Enters $2 + 3i$.
1 f D	2.0000	Stores number in **C** using index 1.

Keystrokes	Display	
7 [ENTER] 4 [f] [I]	7.0000	Enters $7 + 4i$.
2 [f] [D]	7.0000	Stores number in **C** using index 2.
1 [f] [E]	2.0000	Recalls first number.
2 [f] [E]	7.0000	Recalls second number.
[+]	9.0000	Real part of sum.
[f] [Re≷Im]	7.0000	Imaginary part of sum.

Calculating the *n*th Roots of a Complex Number

This program calculates the nth roots of a complex number. The roots are z_k for $k = 0, 1, 2, ..., n - 1$. You can also use the program to calculate $z^{1/r}$, where r isn't necessarily an integer. The program operates the same way except that there may be infinitely many roots z_k for $k = 0, \pm 1, \pm 2, ...$.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000–	
[f] [LBL] [A]	001–42,21,11	
[x≷y]	002– 34	Places n in X-register, z in Y-registers.
[1/x]	003– 15	Calculates $1/n$.
[g] [LSTx]	004– 43 36	Retrieves n.
[R↓]	005– 33	
[g] [SF] 8	006–43, 4, 8	Activiates Complex mode.
[y^x]	007– 14	Calculates $z^{1/n}$.
[STO] 2	008– 44 2	Stores real part of z_0 in R_2.
[f] [Re≷Im]	009– 42 30	
[STO] 3	010– 44 3	Stores imaginary part of z_0 in R_3.
3	011– 3	
6	012– 6	
0	013– 0	
[g] [R↑]	014– 43 33	
[÷]	015– 10	Calculates $360/n$.
[STO] 4	016– 44 4	Stores $360/n$ in R_4.
0	017– 0	
[STO] [I]	018– 44 25	Stores 0 in Index register.

Keystrokes	Display	
f LBL 0	019-42,21, 0	
RCL 4	020- 45 4	Recalls $360/n$.
RCL × I	021-45,20,25	Calculates $360k/n$ using Index register.
f Re≷Im	022- 42 30	
g CLx	023- 43 35	
1	024- 1	Places $1 + i(k360/n)$ in the X-register.
g DEG	025- 43 7	Sets Degrees mode.
f →R	026- 42 1	Calculates $e^{ik360/n}$.
RCL 2	027- 45 2	Recalls real part of z_0.
RCL 3	028- 45 3	Recalls imaginary part of z_0.
f I	029- 42 25	Forms complex z_0.
×	030- 20	Calculates $z_0 e^{ik360/n}$, root number k.
RCL I	031- 45 25	Recalls number k.
x≷y	032- 34	Places z_k in X-registers, k in Y-register.
1	033- 1	
STO + I	034-44,40,25	Increments number k in Index register.
R↓	035- 33	Restores z_k and k to X- and Y-registers.
R/S	036- 31	Halts execution.
GTO 0	037- 22 0	Branch for next root.

Labels used: A and 0.

Registers used: R_2, R_3, R_4, and Index register.

To use this program:

1. Enter the order n into the Y-register and the complex number z into the X-registers.
2. Press f A to calculate the principal root, z_0, which is placed in the real and imaginary X-registers. (Press and hold f (i) to view the imaginary part.)

3. To calculate higher number roots z_k:
   - Press [R/S] to calculate each successive higher-number root. Each root z_k is placed in the complex X-registers and its number k is placed in the Y-register. Between root calculations, you can perform other calculations without disturbing this program (if R_2, R_3, R_4, and the Index register aren't changed).
   - Store the number of the root k in the Index register (using [STO] [I]), then press [R/S] to calculate z_k. The complex root and its number are placed in the X- and Y-registers, respectively. (By pressing [R/S] again, you can continue calculating higher-number roots.)

Example: Use the previous program to compute $(1)^{1/100}$. Calculate z_0, z_1, and z_{50} for this expression.

Keystrokes	Display	
[g] [P/R]		Run mode.
100 [ENTER] 1	**1**	Enters $n = 100$ and $z = 1$ (purely real).
[f] [A]	**1.0000**	Calculates z_0 (real part).
[f] [(i)] (hold)	**0.0000**	Imaginary part of z_0.
[R/S]	**0.9980**	Calculates z_1 (real part).
[f] [(i)] (hold)	**0.0628**	Imaginary part of z_1.
50 [STO] [I]	**50.0000**	Stores root number in Index register.
[R/S]	**-1.0000**	Calculates z_{50} (real part).
[f] [(i)] (hold)	**0.0000**	Imaginary part of z_{50}.

Solving an Equation for Its Complex Roots

A common method for solving the complex equation $f(z) = 0$ numerically is Newton's iteration. This method starts with an approximation z_0 to a root and repeatedly calculates

$$z_{k+1} = z_k - f(z_k)/f'(z_k)$$

until z_k converges.

The following example shows how [SOLVE] can be used with Newton's iteration to estimate complex roots. (A different

technique that doesn't use Complex mode is mentioned on page 16.)

Example: The response of an automatically controlled system to small transient perturbations has been modeled by the differential-delay equation

$$\frac{d}{dt}w(t) + 9\,w(t) + 8\,w(t-1) = 0.$$

How stable is this system? In other words, how rapidly do solutions of this equation decay?

Every solution $w(t)$ is known to be expressible as a sum

$$w(t) = \sum_k c(z)e^{zt}$$

involving constant coefficients $c(z)$ chosen for each root z of the differential-delay equation's associated characteristic equation:

$$z + 9 + 8e^{-z} = 0.$$

Every root $z = x + iy$ contributes to $w(t)$ a component $e^{zt} = e^{xt}(\cos(yt) + i\,\sin(yt))$ whose rate of decay is faster as x, the real part of z, is more negative. Therefore, the answer to the question entails the calculation of all the roots z of the characteristic equation. Since that equation has infinitely many roots, none of them real, the calculation of *all* roots could be a large task.

However, the roots z are known to be approximated for large integers n by $z \approx A(n) = -\ln((2n + \frac{1}{2})\pi/8) \pm i(2n + \frac{1}{2})\pi$ for $n = 0, 1, 2, \ldots$. The bigger is n, the better is the approximation. Therefore you need calculate only the few roots not well approximated by $A(n)$—the roots with $|z|$ not very big.

When using Newton's iteration, what should $f(z)$ be for this problem? The obvious function $f(z) = z + 9 + 8e^{-z}$ isn't a good choice because the exponential grows rapidly for larger negative values of $\mathrm{Re}(z)$. This would slow convergence considerably unless the first guess z_0 were extremely close to a root. In addition, this $f(z)$ vanishes infinitely often, so it's difficult to determine when all desired roots have been calculated. But by rewriting this equation as

$$e^z = -8/(z + 9)$$

and taking logarithms, you obtain an equivalent equation

$$z = \ln(-8/(z+9)) \pm i2n\pi \qquad \text{for } n = 0, 1, 2, \ldots$$

This equation has only two complex conjugate roots z for each integer n. Therefore use the equivalent function

$$f(z) = z - \ln(-8/(z+9)) \pm i2n\pi \qquad \text{for } n = 0, 1, 2, \ldots.$$

and apply Newton's iteration

$$z_{k+1} = z_k - (z_k - \ln(-8/(z_k+9)) \pm i2n\pi)/(1 + 1/(z_k+9)).$$

As a first guess, choose z_0 as $A(n)$, the approximation given earlier. A bit of algebraic rearrangement using the fact that $\ln(\pm i) = \pm i\pi/2$ leads to this formula:

$$z_{k+1} = A(n) + ((z_k - A(n)) + (z_k+9)\ln(i\mathrm{Im}(A(n))/(z_k+9)))/(z_k+10).$$

In the program below, $\mathrm{Re}(A(n))$ is stored in R_0 and $\mathrm{Im}(A(n))$ is stored in R_1. Note that only one of each conjugate pair of roots is calculated for each n.

Keystrokes	Display		Comments
[g] [P/R]			Program mode.
[f] CLEAR [PRGM]	000-		
[f] [LBL] [A]	001-	42,21,11	Program for $A(n)$.
[g] [CF] 8	002-	43, 5, 8	Specifies real arithmetic.
[ENTER]	003-	36	
[+]	004-	40	
[.]	005-	48	
5	006-	5	
[+]	007-	40	
[g] [π]	008-	43 26	
[×]	009-	20	Calculates $(2n + ½)\pi$.
[ENTER]	010-	36	
[STO] 1	011-	44 1	
8	012-	8	
[÷]	013-	10	
[g] [LN]	014-	43 12	
[CHS]	015-	16	Calculates $-\ln((2n + ½)\pi/8)$.
[STO] 0	016-	44 0	
[x≷y]	017-	34	
[f] [I]	018-	42 25	Forms complex $A(n)$.

Keystrokes	Display	
[g] [RTN]	019– 43 32	
[f] [LBL] [B]	020–42,21,12	Program for z_{k+1}.
[ENTER]	021– 36	
[ENTER]	022– 36	
[RCL] 1	023– 45 1	
[f] [Re≷Im]	024– 42 30	Creates $i\text{Im}(A(n))$.
[x≷y]	025– 34	
9	026– 9	
[+]	027– 40	
[÷]	028– 10	
[g] [LSTx]	029– 43 36	
[x≷y]	030– 34	
[g] [LN]	031– 43 12	
[×]	032– 20	
[x≷y]	033– 34	
[RCL] 1	034– 45 1	
[f] [Re≷Im]	035– 42 30	
[RCL] [+] 0	036–45,40, 0	
[−]	037– 30	
[g] [LSTx]	038– 43 36	
[R↓]	039– 33	
[+]	040– 40	
[x≷y]	041– 34	
1	042– 1	
0	043– 0	
[+]	044– 40	
[÷]	045– 10	
[+]	046– 40	
[g] [RTN]	047– 43 32	
[f] [LBL] [C]	048–42,21,13	Program for residual, $\lvert e^z + 8/(z+9)\rvert$.
[ENTER]	049– 36	
[e^x]	050– 12	
9	051– 9	
[g] [LSTx]	052– 43 36	
[+]	053– 40	
8	054– 8	
[x≷y]	055– 34	
[÷]	056– 10	
[+]	057– 40	

Keystrokes	Display		
g ABS	058-	43 16	Calculates $\lvert e^z + 8/(z+9)\rvert$.
g RTN	059-	43 32	

Labels used: A, B, and C.

Registers used: R_0 and R_1.

Now run the program. For each root, press B until the displayed real part doesn't change. (You might also check that the imaginary part doesn't change.)

Keystrokes	Display		
g P/R			Run mode.
f USER			Activates User mode.
0 A	1.6279		Displays $\mathrm{Re}(A(0)) = \mathrm{Re}(z_0)$.
B	-0.1487		$\mathrm{Re}(z_1)$.
B	-0.1497		$\mathrm{Re}(z_2)$.
B	-0.1497		$\mathrm{Re}(z)$.
f (i) (hold)	2.8319		$\mathrm{Im}(z)$.
C	1.0000	-10	Calculates residual.
x≷y	-0.1497		Restores z to X-register.

By repeating the same process for $n = 1$ through 5, you will obtain the results listed below. (Only one of each pair of complex roots is listed.)

n	$A(n)$	Root z_k	Residual
0	$1.6279 + i1.5708$	$-0.1497 + i2.8319$	1×10^{-10}
1	$0.0184 + i7.8540$	$-0.4198 + i8.6361$	6×10^{-10}
2	$-0.5694 + i14.1372$	$-0.7430 + i14.6504$	2×10^{-9}
3	$-0.9371 + i20.4204$	$-1.0236 + i20.7868$	5×10^{-10}
4	$-1.2054 + i26.7035$	$-1.2553 + i26.9830$	9×10^{-10}
5	$-1.4167 + i32.9867$	$-1.4486 + i33.2103$	2×10^{-9}

As n increases, the first guess $A(n)$ comes ever closer to the desired root z. (When you're finished, press [f] [USER] to deactivate User mode.)

Since all roots have negative real parts, the system is stable, but the margin of stability (the smallest in magnitude among the real parts, namely -0.1497) is small enough to cause concern if the system must withstand much noise.

Contour Integrals

You can use [∫] to evaluate the contour integral $\int_C f(z)dz$, where C is a curve in the complex plane.

First parameterize the curve C by $z(t) = x(t) + iy(t)$ for $t_1 \leq t \leq t_2$. Let $G(t) = f(z(t))z'(t)$. Then

$$\int_C f(z)dz = \int_{t_1}^{t_2} G(t)dt$$

$$= \int_{t_1}^{t_2} \mathrm{Re}(G(t))dt + i\int_{t_1}^{t_2} \mathrm{Im}(G(t))dt.$$

These integrals are precisely the type that [∫] evaluates in Complex mode. Since $G(t)$ is a complex function of a real variable t, [∫] will sample $G(t)$ on the interval $t_1 \leq t \leq t_2$ and integrate $\mathrm{Re}(G(t))$—the value that your function returns to the real X-register. For the imaginary part, integrate a function that evaluates $G(t)$ and uses [Re ≷ Im] to place $\mathrm{Im}(G(t))$ into the real X-register.

The general-purpose program listed below evaluates the complex integral

$$I = \int_a^b f(z)dz$$

along the straight line from a to b, where a and b are complex numbers. The program assumes that your complex function subroutine is labeled "B" and evaluates the complex function $f(z)$, and that the limits a and b are in the complex Y- and X-registers, respectively. The complex components of the integral I and the uncertainty ΔI are returned in the X- and Y-registers.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	

Keystrokes	Display	
f LBL A	001-42,21,11	
x≷y	002- 34	
−	003- 30	Calculates $b - a$.
STO 4	004- 44 4	Stores Re$(b-a)$ in R_4.
f Re≷Im	005- 42 30	
STO 5	006- 44 5	Stores Im$(b-a)$ in R_5.
g LSTx	007- 43 36	Recalls a.
STO 6	008- 44 6	Stores Re(a) in R_6.
f Re≷Im	009- 42 30	
STO 7	010- 44 7	Stores Im(a) in R_7.
0	011- 0	
ENTER	012- 36	
1	013- 1	
f ß 0	014-42,20, 0	Calculates Im(I) and Im(ΔI).
STO 2	015- 44 2	Stores Im(I) in R_2.
R↓	016- 33	
STO 3	017- 44 3	Stores Im(ΔI) in R_3.
R↓	018- 33	
f ß 1	019-42,20, 1	Calculates Re(I) and Re(ΔI).
RCL 2	020- 45 2	Recalls Im(I).
f I	021- 42 25	Forms complex I.
x≷y	022- 34	
RCL 3	023- 45 3	Recalls Im(ΔI).
f I	024- 42 25	Forms complex ΔI.
x≷y	025- 34	Restores I to X-register.
g RTN	026- 43 32	
f LBL 0	027-42,21, 0	Subroutine for Im$(f(z)z'(t))$.
GSB 1	028- 32 1	Calculates $f(z)z'(t)$.
f Re≷Im	029- 42 30	Swaps complex components.
g RTN	030- 43 32	
f LBL 1	031-42,21, 1	Subroutine to calculate $f(z)z'(t)$.

Keystrokes	Display		
RCL 4	032–	45 4	
RCL 5	033–	45 5	
f I	034–	42 25	Forms complex $b-a$.
×	035–	20	Calculates $(b-a)t$.
RCL 6	036–	45 6	
RCL 7	037–	45 7	
f I	038–	42 25	Forms complex a.
+	039–	40	Calculates $a+(b-a)t$.
GSB B	040–	32 12	Calculates $f(a+(b-a)t)$.
RCL 4	041–	45 4	
RCL 5	042–	45 5	
f I	043–	42 25	Forms complex $z'(t)=b-a$.
×	044–	20	Calculates $f(z)z'(t)$.
g RTN	045–	43 32	

Labels used: A, 0, and 1.

Registers used: R_2, R_3, R_4, R_5, R_6, and R_7.

To use this program:

1. Enter your function subroutine labeled "B" into program memory.
2. Press 7 f DIM (i) to reserve registers R_0 through R_7. (Your subroutine may require additional registers.)
3. Set the display format for f.
4. Enter the two complex points that define the ends of the straight line that your function will be integrated along. The lower limit should be in the Y-registers; the upper limit should be in the X-registers.
5. Press f A to calculate the complex line integral. The value of the integral is in the X-registers; the value of the uncertainty is in the Y-registers.

Because two integrals are being evaluated, the program will usually take longer than a real integral, although the f routine doesn't have to use the same number of sample points for both integrals. The easier integral will use less calculations than the more difficult one.

Example: Approximate the integrals

$$I_1 = \int_1^{\infty} \frac{\cos x}{x + 1/x}\,dx \qquad \text{and} \qquad I_2 = \int_1^{\infty} \frac{\sin x}{x + 1/x}\,dx.$$

These integrands decay very slowly as x approaches infinity and therefore require a long interval of integration and a long execution time. You can expedite this calculation by deforming the path of integration from the real axis into the complex plane. According to complex variable theory, these integrals can be combined as

$$I_1 + iI_2 = \int_1^{1+i\infty} \frac{e^{iz}}{z + 1/z}\,dz.$$

This complex integrand, evaluated along the line $x = 1$ and $y \geq 0$, decays rapidly as y increases—like e^{-y}.

To use the previous program to calculate both integrals at the same time, write a subroutine to evaluate

$$f(z) = \frac{e^{iz}}{z + 1/z}.$$

Keystrokes	Display	
f LBL B	046-42,21,12	
1/x	047- 15	
g LSTx	048- 43 36	
+	049- 40	Calculates $z + 1/z$.
g LSTx	050- 43 36	
1	051- 1	
f Re≷Im	052- 42 30	Forms $0 + i$.
×	053- 20	
e^x	054- 12	Calculates e^{iz}.
x≷y	055- 34	
÷	056- 10	Calculates $f(z)$.
g RTN	057- 43 32	

Approximate the complex integral by integrating the function from $1 + 0i$ to $1 + 6i$ using a SCI 2 display format to obtain three significant digits. (The integral beyond $1 + 6i$ doesn't affect the first three digits.)

Keystrokes	Display		
g P/R			Run mode.
f SCI 2			Specifies SCI 2 format.
1 ENTER	1.00	00	Enters first limit of integration, $1 + 0i$.
1 ENTER 6	6		
f I	1.00	00	Enters second limit of integration, $1 + 6i$.
f A	-3.24	-01	Calculates I and displays $\mathrm{Re}(I) = I_1$ (after about 9 minutes).
f (i) (hold)	3.82	-01	Displays $\mathrm{Im}(I) = I_2$.
x≷y	7.87	-04	Displays $\mathrm{Re}(\Delta I) = \Delta I_1$.
f (i) (hold)	1.23	-03	Displays $\mathrm{Im}(\Delta I) = \Delta I_2$.
f FIX 4	0.0008		

This result I is calculated much more quickly than if I_1 and I_2 were calculated directly along the real axis.

Complex Potentials

Conformal mapping is useful in applications associated with a complex potential function. The discussion that follows deals with the problem of fluid flow, although problems in electrostatics and heat flow are analagous.

Consider the potential function $P(z)$. The equation $\mathrm{Im}(P(z)) = c$ defines a family of curves that are called *streamlines* of the flow. That is, for any value of c, all values of z that satisfy the equation lie on a streamline corresponding to that value of c. To calculate some points z_k on the streamline, specify some values for x_k and then use SOLVE to find the corresponding values of y_k using the equation

$$\mathrm{Im}(P(x_k + iy_k)) = c.$$

If the x_k values are not too far apart, you can use y_{k-1} as an initial estimate for y_k. In this way, you can work along the streamline and calculate the complex points $z_k = x_k + iy_k$. Using a similar procedure, you can define the equipotential lines, which are given by $\mathrm{Re}(P(z)) = c$.

The program listed below is set up to compute the values of y_k from evenly spaced values of x_k. You must provide a subroutine labeled "B" that places Im($P(z)$) in the real X-register. The program uses inputs that specify the step size h, the number of points n along the real axis, and $z_0 = x_0 + iy_0$, the initial point on the streamline. You must enter n, h, and z_0 into the Z-, Y-, and X-registers before running the program.

The program computes the values of z_k and stores them in matrix **A** in the form $a_{k1} = x_{k-1}$ and $a_{k2} = y_{k-1}$ for $k = 1, 2, ..., n$.

Keystrokes	Display	
[g] [P/R]		Program mode.
[f] CLEAR [PRGM]	000-	
[f] [LBL] [A]	001-42,21,11	
[R↓]	002- 33	
[STO] 4	003- 44 4	Stores h in R_4.
[R↓]	004- 33	
2	005- 2	
[f] [DIM] [A]	006-42,23,11	Dimensions matrix **A** to be $n \times 2$.
[g] [CLx]	007- 43 35	
[STO] [MATRIX] [A]	008-44,16,11	Makes all elements of **A** be zero.
[STO] [I]	009- 44 25	Stores zero in Index register.
[f] [MATRIX] 1	010-42,16, 1	Sets $R_0 = R_1 = 1$.
[g] [R↑]	011- 43 33	Recalls z_0 to X-registers.
[STO] 2	012- 44 2	Stores x_0 in R_2.
[f] [USER] [STO] [A]	013u 44 11	Sets $a_{11} = x_0$.
[f] [USER]		
[f] [Re≷Im]	014- 42 30	
[STO] 3	015- 44 3	Stores y_0 in R_3.
[f] [USER] [STO] [A]	016u 44 11	Sets $a_{12} = y_0$.
[f] [USER]		
[GTO] 1	017- 22 1	Branches if matrix **A** not full ($n > 1$).
[f] [LBL] 0	018-42,21, 0	
[RCL] [MATRIX] [A]	019-45,16,11	Recalls descriptor of matrix **A**.

Keystrokes	Display	
g RTN	020- 43 32	
f LBL 1	021-42,21, 1	
f Re≷Im	022- 42 30	Restores z_0.
GSB B	023- 32 12	Calculates Im($P(z_0)$) (or Re($P(z_0)$)) for equipotential line.)
STO 5	024- 44 5	Stores c in R_5.
f LBL 2	025-42,21, 2	Loop for finding y_k.
1	026- 1	
STO + I	027-44,40,25	Increments counter k in Index register.
RCL 4	028- 45 4	Recalls h.
RCL I	029- 45 25	Recalls counter k.
×	030- 20	Calculates kh.
RCL 2	031- 45 2	Recalls x_0.
+	032- 40	Calculates $x_k = x_0 + kh$.
STO 6	033- 44 6	Stores x_k in R_6.
RCL 3	034- 45 3	Recalls y_{k-1} from R_3.
ENTER	035- 36	Duplicates y_{k-1} for second estimate.
f SOLVE 3	036-42,10, 3	Searches for y_k.
GTO 4	037- 22 4	Branches for valid y_k root.
1	038- 1	Starts decreasing step size.
STO - I	039-44,30,25	Decrements counter k.
4	040- 4	
STO ÷ 4	041-44,10, 4	Reduces h by factor of 4.
STO × I	042-44,20,25	Multiplies counter by 4.
GTO 2	043- 22 2	Loops back to find y_k again.
f LBL 4	044-42,21, 4	Continues finding y_k.
RCL 6	045- 45 6	
f PSE	046- 42 31	Displays x_k.
f USER STO A	047u 44 11	Sets $a_{k+1,1} = x_k$.
f USER		

Keystrokes	Display	
R↓	048- 33	
f PSE	049- 42 31	Displays y_k.
STO 3	050- 44 3	Stores y_k in R_3.
f USER STO A	051u 44 11	Sets $a_{k+1,2} = y_k$.
f USER		
GTO 2	052- 22 2	Branch for $k+1 < n$ (**A** isn't full).
GTO 0	053- 22 0	Branch for $k+1 = n$ (**A** is full).
f LBL 3	054-42,21, 3	Function subroutine for SOLVE.
RCL 6	055- 45 6	Recalls x_k.
x≷y	056- 34	Restores current estimate for y_k.
f I	057- 42 25	Creates estimate $z_k = x_k + iy_k$.
GSB B	058- 32 12	Calculates $\mathrm{Im}(P(z_k))$ (or $\mathrm{Re}(P(z_k))$ for equipotential lines).
RCL 5	059- 45 5	Recalls c.
-	060- 30	Calculates $\mathrm{Im}(P(z_k)) - c$.
g RTN	061- 43 32	

Labels used: A, B, 0, 1, 2, 3, and 4.

Registers used: R_0, R_1, R_2 (x_0), R_3 (y_0), R_4 (h), R_5 (c), R_6 (x_k), and Index register (k).

Matrix used: **A**.

One special feature of this program is that if an x_k value lies beyond the domain of the streamline (so that there is no root for SOLVE to find), then the step size is decreased so that x_k approaches the boundary where the streamline turns back. This feature is useful for determining the nature of the streamline when y_k isn't a single-valued function of x_k. If h is small enough, the values of z_k will lie on one branch of the streamline and approach the boundary. (The second example below illustrates this feature.)

To use this program:

1. Enter your subroutine labeled "B" into program memory. It should place into the real X-register $\mathrm{Im}(P(z))$ when calculating streamlines or $\mathrm{Re}(P(z))$ when calculating equipotential lines.
2. Press 6 [f] [DIM] [(i)] to reserve registers R_0 through R_6 (and the Index register). (Your subroutine may require additional registers.)
3. Enter the values of n and h into the Z- and Y-registers by pressing n [ENTER] h [ENTER].
4. Enter the complex value of $z_0 = x_0 + iy_0$ into the X-registers by pressing x_0 [ENTER] y_0 [f] [I].
5. Press [f] [A] to display the successive values of x_k and y_k for $k = 1, \ldots, n$ and finally the descriptor of matrix **A**. The values for $k = 0, \ldots, n$ are stored in matrix **A**.
6. If desired, recall values from matrix **A**.

Example: Calculate the streamline of the potential $P(z) = 1/z + z$ passing through $z = -2 + 0.1i$.

First, enter subroutine "B" to compute $\mathrm{Im}(P(z))$.

Keystrokes	Display	
[f] [LBL] [B]	062-42,21,12	
[ENTER]	063- 36	Duplicates z.
[1/x]	064- 15	
[+]	065- 40	Calculates $1/z + z$.
[f] [Re≷Im]	066- 42 30	Places $\mathrm{Im}(P(z))$ in X-register.
[g] [RTN]	067- 43 32	

Determine the streamline using $z_0 = -2 + 0.1i$, step size $h = 0.5$, and number of points $n = 9$.

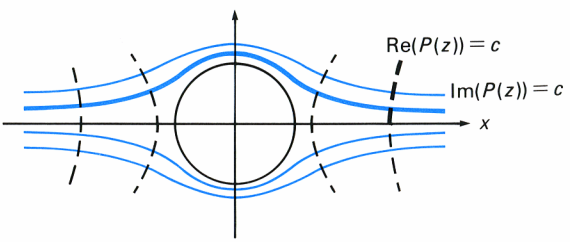
Keystrokes	Display	
[g] [P/R]		Run mode.
9 [ENTER]	9.0000	Enters n.
.5 [ENTER]	0.5000	Enters h.

Keystrokes	Display			
2 [CHS] [ENTER]	-2.0000			
.1 [f] [I]	-2.0000			Enters z_0.
[f] [A]	-1.5000			x_1.
	0.1343			y_1.
	⋮			⋮
	2.0000			x_9.
	0.1000			y_9.
	A	9	2	Descriptor for matrix **A**.
[g] [CF] 8	A	9	2	Deactivates Complex mode.

Matrix **A** contains the following values of x_k and y_k:

x_k	y_k
-2.0	0.1000
-1.5	0.1343
-1.0	0.4484
-0.5	0.9161
0.0	1.0382
0.5	0.9161
1.0	0.4484
1.5	0.1343
2.0	0.1000

The streamline and velocity equipotential lines are illustrated below. The derived streamline corresponds to the heavier solid line.



Example: For the same potential as the previous example, $P(z) = 1/z + z$, compute the velocity equipotential line starting at $z = 2 + i$ and proceeding to the left.

First change subroutine "B" so that it returns Re($P(z)$)—that is, remove the [Re ↔ Im] instruction from "B". Try $n = 6$ and $h = -0.5$. (Notice that h is negative, which specifies that x_k will be to the left of x_0.)

Although the keystrokes are not listed here, the results that would be calculated and stored in matrix **A** are shown below.

x_k	y_k
2.0000	1.0000
1.8750	0.2362
1.8672	0.1342
1.8652	0.0941
1.8647	0.0811
1.8646	0.0775

The results show the nature of the top branch of the curve (the heavier dashed line in the graph for the previous example). Note that the step size h is automatically decreased in order to follow the curve—rather than stop with an error—when no y-value is found for $x < 1.86$.

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

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

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

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

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

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

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

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

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

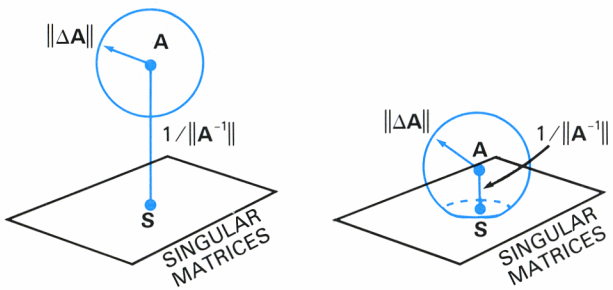
$$\Delta\mathbf{A} = \begin{bmatrix} 0 & -5 \times 10^{-11} \\ 0 & 5 \times 10^{-11} \end{bmatrix}$$

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 $\mathbf{A}$ and matrix $\mathbf{S}$ 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 $\mathbf{A}$ is a region of matrices that are practically indistinguishable from $\mathbf{A}$ (for example, those within rounding errors of $\mathbf{A}$). The radius of this region is $\|\Delta\mathbf{A}\|$. The distance from a nonsingular matrix $\mathbf{A}$ to the nearest singular matrix $\mathbf{S}$ 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 $\mathbf{A}$, and it may not even be reasonable to try to compute the inverse of $\mathbf{A}$.

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 $\|\Delta\mathbf{A}\| \leq \epsilon\|\mathbf{A}\|$, 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 $\|\mathbf{R}\| \leq \epsilon\|\mathbf{A}\|\|\mathbf{Z}\|$. 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:

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

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

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

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

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

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

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

In order for $\mathbf{F} + \mathbf{Z}$ to be a better approximation to $\mathbf{X}$ than is $\mathbf{Z}$, 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 $\mathbf{A}$ is used for finding both solutions, $\mathbf{Z}$ and $\mathbf{F}$. The LU decomposition formed while calculating $\mathbf{Z}$ can be used for calculating $\mathbf{F}$, 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 $\mathbf{E}$ is ill-conditioned (nearly singular). Even iterative refinement can fail to improve the calculated solution when $\mathbf{E}$ 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 $\mathbf{E}$ is obtained from a matrix $\mathbf{A}$ by $\mathbf{E} = \mathbf{LAR}$, where $\mathbf{L}$ and $\mathbf{R}$ are scaling diagonal matrices whose diagonal elements are all integer powers of 10. Then $\mathbf{E}$ is said to be obtained from $\mathbf{A}$ by *scaling*. $\mathbf{L}$ scales the rows of $\mathbf{A}$, and $\mathbf{R}$ 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 $\mathbf{E}$ 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 $\|\mathbf{E} - \mathbf{S}\| / \|\mathbf{E}\| = \frac{1}{3} \times 10^{-40}$. This means that $K(\mathbf{S}) \geqslant 3 \times 10^{40}$, so it's not surprising that the calculated $\mathbf{E}^{-1}$

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

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

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

which is correct to 10 digits.

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

You can also apply scaling to solving a system of equations, for example $\mathbf{EX} = \mathbf{B}$, 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 $\mathbf{E}$ and $\mathbf{A}$ has roughly the same norm as every other. Using this new system, the HP-15C calculates the solution

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

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

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

Least-Squares Calculations

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

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

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

Normal Equations

From the expression above,

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

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

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

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 $\mathbf{U}$ and the product $\mathbf{Qy}$ 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 $\mathbf{V}$, 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 $\mathbf{V}$ need to be retained. (Note that $\mathbf{Q}$ here is not the same as that mentioned earlier, since this $\mathbf{Q}$ must also transform $\mathbf{y}$.)

2. Solve the following system for **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

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

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

Singular and Nearly Singular Matrices

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

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

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

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

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

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

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

since they both have identical calculated LU decompositions.

On the other hand, the matrix

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

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

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

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

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

since they both have identical calculated LU decompositions.

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

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

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

Applications

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

Constructing an Identity Matrix

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

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

Labels used: 8 and 9.

Registers used: R_0, R_1, and Index register.

One-Step Residual Correction

The following program solves the system of equations $\mathbf{AX} = \mathbf{B}$ for $\mathbf{X}$, 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.

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

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

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

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

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

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

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

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 $x_2^{(1)}$

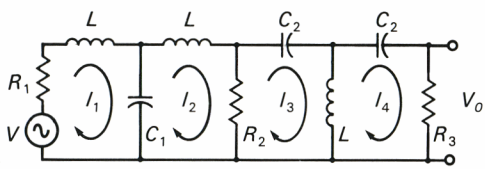
By repeating the last four steps, you will obtain these results:

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 $P_{y,x}$	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 $P_{y,x}$	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 $C_{y,x}$	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).
$x \gtrless y$ g →P	4.1013		-03	Displays $\lvert I_1 \rvert$.
$x \gtrless y$	8.7212		01	Displays Arg(I_1) in degrees.
RCL C	-1.4489		-03	
RCL C	-3.5633		-02	
$x \gtrless y$ g →P	3.5662		-02	Displays $\lvert I_2 \rvert$.
$x \gtrless y$	-9.2328		01	
RCL C	-1.4541		-03	
RCL C	-3.5633		-02	
$x \gtrless y$ 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 $n \geq p$ and $(n+p)(p+1) \leq 56$. 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 $n \geq p$ and $(n+p)(p+1) \leq 58$. 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 $\mathbf{D} = \mathbf{A}^T\mathbf{B}$.
[$x \rightleftarrows y$]	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

$$\begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & & & s & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & \ddots & & \\ & \mathbf{0} & & & & & & & \\ & & & & & & & 1 & \\ & & & -s & & & & c & \end{bmatrix}$$

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 $x \leq y$	059- 43 10	Tests $p+2 \leq k$.
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 $\mathbf{A}$ 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 $\mathbf{Q}$ 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 $\mathbf{A}$ 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 $\mathbf{A}$, 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 $\mathbf{A}$; the smaller are all the elements of $\mathbf{B}$, the better the approximation. Therefore $\mathbf{Q}^T\mathbf{A}\mathbf{Q}$ must be more nearly diagonal than $\mathbf{A}$ 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 R→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?

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

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

$$= 7$$

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 $\mathbf{P}^{-1}\mathbf{AP} = \text{diag}(\lambda_1,\lambda_2,\ldots)$.

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	021u 44 13	Stores random components in **C**.
f USER		
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	034u 45 13	
f USER		
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 LST*x*	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 $\mathbf{A}$, return the gradient in matrix $\mathbf{E}$, 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 $\mathbf{A}$.

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 $\mathbf{E}$.
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 $x \leq y$	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 $\mathbf{x}$ 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 $\nabla v(w,h) = \mathbf{0}$.)

Appendix

Accuracy of Numerical Calculations

Misconceptions About Errors

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

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

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

should equal x too.

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

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

which seems very wrong. Should this calculator be repaired?

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

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

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

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

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

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

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

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

Example 3: Pi. The constant $\pi = 3.14159265358979323846264338...$. Pressing the [π] key on the HP-15C delivers a different value

$$[\pi] = 3.141592654$$

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

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

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

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

The measured lengths turn out to be respectively

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

The discrepancy in total length is

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

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

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

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

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

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

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

and

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

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

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

and

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

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

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

If $0 < x < 1$, then $x \leqslant 0.9999999999$ in a 10-digit calculator. We would then rightly expect that $\sqrt{x} \leqslant \sqrt{0.9999999999}$, which is $0.999999999949999999998\ldots$, which rounds to 0.9999999999 again. Therefore, if $\boxed{\sqrt{x}}$ is pressed arbitrarily often starting with $x < 1$, the result cannot exceed 0.9999999999 . This explains why we obtain $R(x) = 0.9999999999$ for $0 < x < 1$ above. When $R(x)$ is squared 50 times to produce $F(x) = S(R(x))$, the result is clearly 1 for $x \geqslant 1$, but why is $F(x) = 0$ for $0 \leqslant x < 1$? When $x < 1$,

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

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

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

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

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

Similarly, the binomial theorem was used for

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

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

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

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

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

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

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

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

$$\cos x = 0.9999999999$$

and

$$1 - \cos x = 0.0000000001$$

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

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

Then

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

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

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

When this latter expression is evaluated (in Radians mode) at $x = 1.2 \times 10^{-5}$, the computed result $C(x) = 0.5$ is correct to 10 significant digits. This example, while explaining the meaning of the word "cancellation," suggests that it is always a bad thing. That is another misconception to be dispatched later. For the

present, recall that example 1 contains no subtraction, therefore no cancellation, and is still devastated by its rounding error. In this respect example 1 is counterintuitive, a little bit scary. Nowhere in it can we find one or two arithmetic operations to blame for the catastrophe; no small rearrangement will set everything right as happened for $c(x)$. Alas, example 1 is not an isolated example. As computers and calculators grow in power, so do instances of insidious error growth become more common.

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

A Hierarchy of Errors

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

Level 0: No Error

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

Examples:

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

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

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

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

Also exact for real argments are [ABS], [FRAC], [INT], [RND], and comparisons (such as [x≤y]). But the matrix functions [×], [÷], [1/x], [MATRIX] 6, and [MATRIX] 9 (determinant) are exceptions (refer to page 192).

Level ∞: Overflow/Underflow

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

Level 1: Correctly Rounded, or Nearly So

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

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

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

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

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

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

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

Example 2 Explained. Susan tried to calculate

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

where

payment = \$0.01,

$i = 0.1125$, and

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

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

$$i/n = 0.000000003567351598$$
$$1 + i/n = 1.000000004$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

from which the correct total follows.

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

Level 1C: Complex Level 1

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

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

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

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

Level 2: Correctly Rounded for Possibly Perturbed Input

Trigonometric Functions of Real Radian Angles

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

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

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

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

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

Consider

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

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

$$10^{14} \times [\pi] = 314159265400000$$

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

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

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

true	$\pi = 3.14159265358979323846264 33...$	
key	$[\pi] = 3.141592654$	(matches π to 10 digits)
internal	$p = 3.141592653590$	(matches π to 13 digits)

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

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

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

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

This formula has important practical implications:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

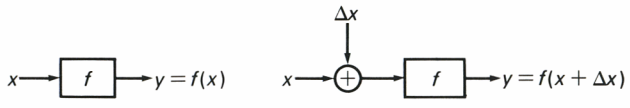
Backward Error Analysis

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

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

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

then in consequence the output $y + \Delta y = f(x + \Delta x)$ would generally be contaminated by noise $\Delta y = f(x + \Delta x) - f(x)$.



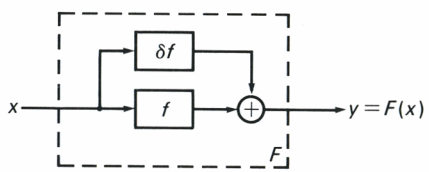
No Noise Noisy Input

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

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

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

where the perturbation δf represents the internal noise in F.



One Small Output Perturbation (Level 1)

We hope the noise term δf is negligible compared with f. When that hope is fulfilled, we classify F in Level 1 for the purposes of

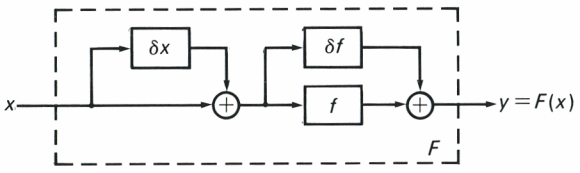
exposition; this means that the noise internal to F can be explained as *one* small addition δf to the intended output f.

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

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

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

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



Small Input and Output Perturbations (Level 2)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Backward Error Analysis Versus Singularities

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

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

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

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

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

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

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

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

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

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

all calculated accurately because [LN] is in Level 1. What might happen if [LN] were in Level 2 instead?

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

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

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

Summary to Here

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

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

A simple example illustrates all four points.

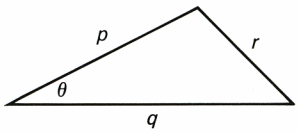
Example 7: The Angle in a Triangle. The cosine law for triangles says

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

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

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

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



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

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

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

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

Disparate Results from Three Programs F_A, F_B, F_C

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

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

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

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

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

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

1. If $p < q$, then swap them to ensure $p \geq q$.
2. Calculate $b = (p - q) + r$, $c = (p - r) + q$, and $s = (p + r) + q$.
3. Calculate
$$a = \begin{cases} r - (p - q) & \text{if } q \geq r \geq 0 \\ q - (p - r) & \text{if } r > q \geq 0 \\ \textbf{Error 0} & \text{otherwise (no triangle exists).} \end{cases}$$
4. Calculate $F_C(p,q,r) = 2\tan^{-1}(\sqrt{ab}/\sqrt{cs})$.

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

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

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

Backward Error Analysis of Matrix Inversion

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

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

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

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

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

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

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

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

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

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

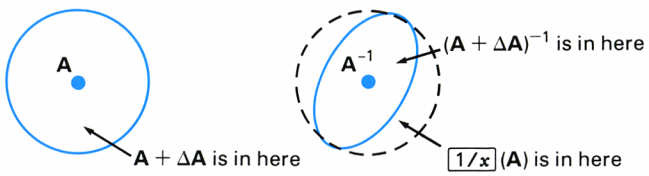
$$\min_{\det(\mathbf{S})=0} \|\mathbf{A} - \mathbf{S}\| = \sigma(\mathbf{A})\,\|\mathbf{A}\|.$$

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

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

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

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



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

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

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

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

If $\mathbf{A}$ is far enough from singularity that all

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

then also

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

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

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

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

and

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

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

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

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

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

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

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

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

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

Despite that

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

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

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

$$\tilde{\mathbf{X}} = \begin{bmatrix} 2. & -5. & 5.000003 & -4.5\times10^{-12} \\ 0 & 5. & -5.000003 & 4.5\times10^{-12} \\ 0 & 0 & 2. & -5.000003 \\ 0 & 0 & 0 & 5.2 \end{bmatrix}$$

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

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

yielded $\mathbf{X}^{-1}$, with $p = q = 0$. The HP-15C calculates $\boxed{1/x}(\tilde{\mathbf{X}}) = \tilde{\mathbf{X}}^{-1}$ except that $q = 0$ is replaced by $q = 9.6\times10^{-11}$, a negligible change. This illustrates how drastically the perceived quality of computed results can be altered by scaling. (Refer to section 4 for more information about scaling.)

Is Backward Error Analysis a Good Idea?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$x = s/a$$

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

The s calculation avoids destructive cancellation.

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

The results of several cases are summarized below.

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

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

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

Case 4: 1 and 0.9999969434

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

Case 6: $12.21711755 \pm i0.001374514$

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

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

Keystrokes	Display
[g] [P/R]	
[f] CLEAR [PRGM]	000-
[f] [LBL] [B]	001-42,21,12
[STO] [I]	002- 44 25
[R↓]	003- 33

*Program "B" exploits a tricky property of the [Σ-] and [Σ+] keys whereby certain calculations can be carried out to 13 significant digits before being rounded back to 10.

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

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

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

Index

Page numbers in **bold** type indicate primary references; page numbers in regular type indicate secondary references.

D

E

H

I

J

L

O

P

Q

R

S

T

U

Corvallis Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

00015-90011 Printed in U.S.A. 8/82