Part III

HP-15C
Advanced Functions

Section 11

Calculating With Complex Numbers

The HP-15C enables you to calculate with complex numbers, that is, numbers of the form

$$a + ib,$$

where a is the real part of the complex number,

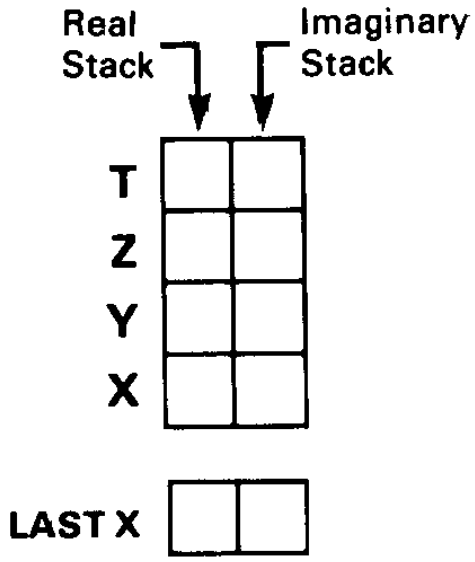
b is the imaginary part of the complex number, and

$i = \sqrt{-1}$.

As you will see, the beauty of calculating with the HP-15C in Complex mode is that once the complex numbers are keyed in, most operations are executed in the same manner as with real numbers.

The Complex Stack and Complex Mode

Calculations with complex numbers are performed using a complex stack composed of *two* parallel four-register stacks (and two LAST X registers). One of these parallel stacks—referred to as the *real* stack—contains the real parts of complex numbers used in calculations. (This is the same stack used in ordinary calculations.) The other stack—referred to as the *imaginary* stack—contains the imaginary parts of complex numbers used in calculations.



Creating the Complex Stack

The imaginary stack is created (by converting five storage registers as described in appendix C) when you activate Complex mode; it does not exist when the calculator is not in Complex mode.

Complex mode is activated

1) automatically, when executing [f][I] or [f][Re≷Im]; or
2) by setting flag 8, the Complex mode flag.

When the calculator is in Complex mode, the **C** annunciator in the display is lit. This tells you that flag 8 is set and the complex stack exists. In or out of Complex mode, *the number appearing in the display is the number in the* **real** *X-register*.

> **Note:** In Complex mode (signified by the **C** annunciator), the HP-15C performs *all* trigonometric functions using *radians*. *The trigonometric mode annunciator in the display* (**RAD**, **GRAD**, *or blank for Degrees*) *applies to two functions only*: [→R] *and* [→P] (as explained later in this section).

Deactivating Complex Mode

Since Complex mode requires the allocation of five registers from memory, you will have more memory available for programming and other advanced functions if you deactivate Complex mode when you are working solely with real numbers.

To deactivate Complex mode, clear flag 8 (keystroke sequence: [g][CF] 8). The **C** annunciator will disappear.

Complex mode is also deactivated when Continuous Memory is reset (as described on page 63). In any case, deactivating Complex mode dissolves the imaginary stack, and all imaginary numbers there are lost.

Complex Numbers and the Stack

Entering Complex Numbers

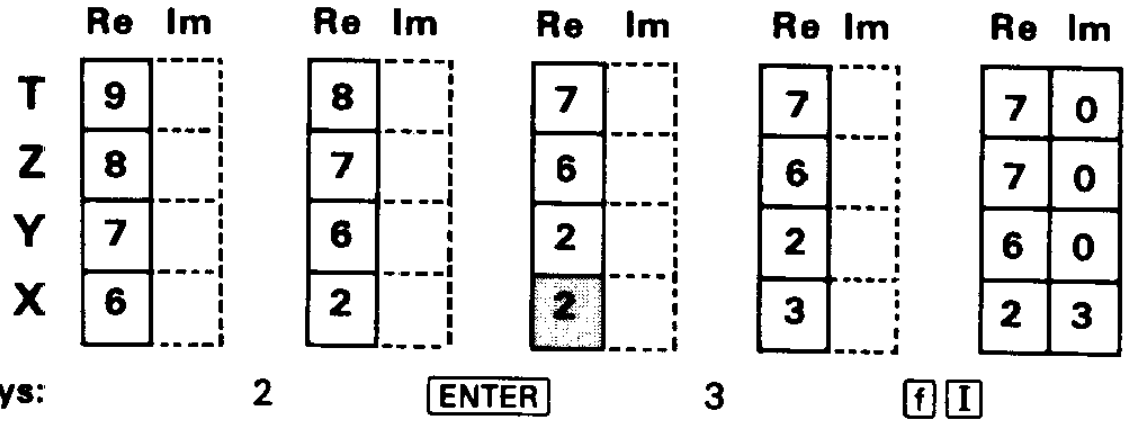
To enter a number with real and imaginary parts:

1. Key the real part of the number into the display.
2. Press [ENTER].
3. Key the imaginary part of the number into the display.
4. Press [f][I]. (If not already in Complex mode, this creates the imaginary stack and displays the **C** annunciator.)

Example: Add $2 + 3i$ and $4 + 5i$. (The operations are illustrated in the stack diagrams following the keystroke listing.)

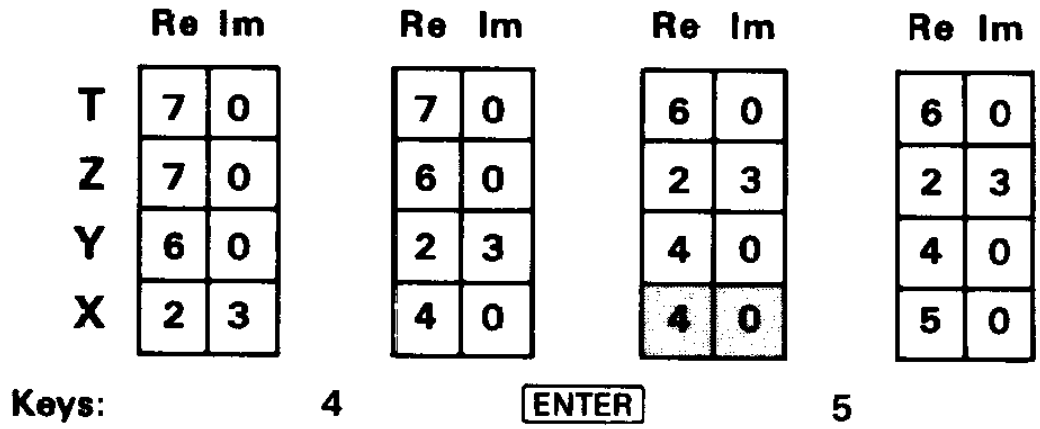
Keystrokes	Display	
f FIX 4		
2 ENTER	2.0000	Keys real part of first number into (real) Y-register.
3	3	Keys imaginary part of first number into (real) X-register.
f I	2.0000	Creates imaginary stack; moves the 3 into the imaginary X-register, and drops the 2 into the real X-register.
4 ENTER	4.0000	Keys real part of second number into (real) Y-register.
5	5	Keys imaginary part of second number into (real) X-register.
f I	4.0000	Copies 5 from real X-register into imaginary X-register, copies 4 from real Y-register into real X-register, and drops stack.
+	6.0000	Real part of sum.
f (i) (hold) (release)	8.0000 6.0000	Displays imaginary part of sum while the (i) key is held. (This also terminates digit entry.)

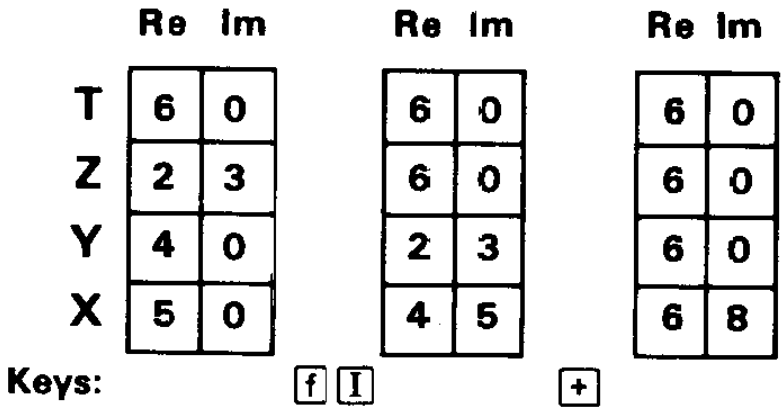
The operation of the real and imaginary stacks during this process is illustrated below. (Assume that the stack registers have been loaded already with the numbers shown as the result of previous calculations.) Note that the imaginary stack, which is shown below at the right of the real stack, is not created until f I is pressed. (Recall also that the shading of the stack indicates that those contents will be written over when the next number is keyed in or recalled.)



The execution of f I causes the entire stack to drop, the T contents to duplicate, and the real X contents to move to the imaginary X-register.

When the second complex number is entered, the stacks operate as shown below. Note that ENTER lifts *both* stacks.





A second method of entering complex numbers is to enter the imaginary part first, then use Re≷Im and ←. This method is illustrated under Entering Complex Numbers With ←, page 127.

Stack Lift in Complex Mode

Stack lift operates on the imaginary stack as it does on the real stack (the real stack behaves identically in and out of Complex mode). *The same functions that enable, disable, or are neutral to lifting of the real stack will enable, disable, or be neutral to lifting of the imaginary stack.* (These processes are explained in detail in section 3 and appendix B.)

In addition, *every nonneutral function except* [←] *and* [CLx] *causes the clearing of the imaginary X-register when the next number is entered.* That is, these functions cause a zero to be placed in the imaginary X-register *when the next number is keyed in or recalled.* Refer to the stack diagrams above for illustrations. This feature allows you to execute calculator operations using the same key sequences you use outside of Complex mode.*

Manipulating the Real and Imaginary Stacks

[Re≷Im] (***real exchange imaginary***). Pressing [f] [Re≷Im] will exchange the contents of the real and imaginary X-registers, thereby converting the imaginary part of the number into the real part and vice-versa. The Y-, Z-, and T-registers are *not* affected. Press [f] [Re≷Im] *twice* to restore a number to its original form.

[Re≷Im] also activates Complex mode if it is not already activated.

Temporary Display of the Imaginary X-Register. Press [f] [(i)] to *momentarily* display the imaginary part of the number in the X-register *without actually switching the real and imaginary parts.* Hold the key down to maintain the display.

Changing Signs

In Complex mode, the [CHS] function affects only the number in the real X-register—the imaginary X-register does not change. This enables you to change the sign of the real or imaginary part without affecting the other. To key in a negative real or imaginary part, change the sign of that part as you enter it.

If you want to find the additive inverse of a complex number *already in the X-register,* however, you cannot simply press [CHS] as you would outside of Complex mode. Instead, you can do either of the following:

---

* Except for the [→P] and [→R] functions, as explained in this section (page 133).

- Multiply by −1.
- If you don't want to disturb the rest of the stack, press [CHS] [f] [Re≷Im] [CHS] [f] [Re≷Im].

To find the negative of only one part of a complex number in the X-register:

- Press [CHS] to negate the *real part only*.
- Press [f] [Re≷Im] [CHS] [f] [Re≷Im] to negate the *imaginary part only*, forming the complex conjugate.

Clearing a Complex Number

Inevitably you will need to clear a complex number. You can clear only one part at a time, but you can then write over both parts (since [←] and [CLx] disable the stack).

Clearing the Real X-Register. Pressing [←] (or [g] [CLx]) with the calculator in Complex mode clears only the number in the real X-register; it does *not* clear the number in the imaginary X-register.

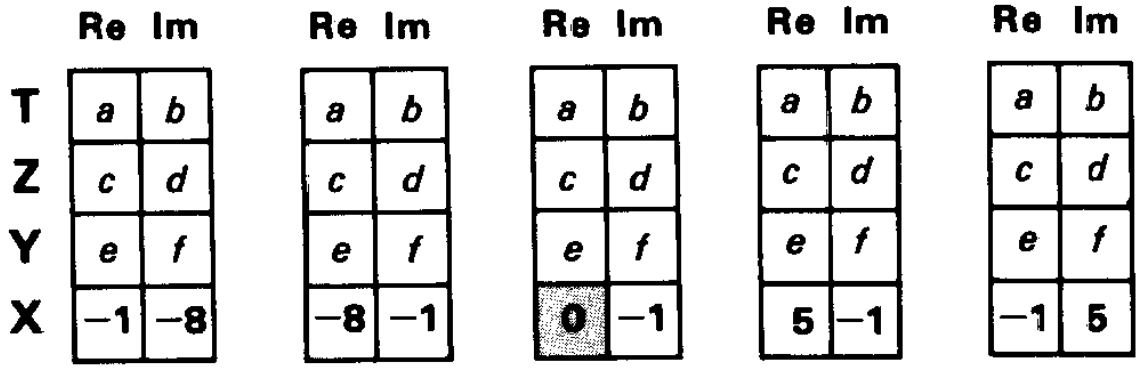
Example: Change $6 + 8i$ to $7 + 8i$ and subtract it from the previous entry. (Use [f] [Re≷Im] or [f] [(i)] to view the imaginary part in X.) Assume a, b, c, and d represent parts of complex numbers.

	Re	Im	Re	Im	Re	Im	Re	Im
T	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*
Z	*c*	*d*	*c*	*d*	*c*	*d*	*a*	*b*
Y	6	0	6	0	6	0	*c*	*d*
X	6	8	0	8	7	8	−1	−8
Keys:			[←]		7		[−] (or other operation)	

Since clearing disables the stack (as explained above), the next number you enter will replace the cleared value. If you want to replace the real part with zero, after clearing use [ENTER] or any other function to terminate digit entry (otherwise the next number you enter will write over the zero); the imaginary part will remain unchanged. You can then continue with any calculator function.

Clearing the Imaginary X-Register. To clear the number in the imaginary X-register, press [f] [Re≷Im], then press [←]. Press [f] [Re≷Im] again to return the zero, or any new number keyed in, to the imaginary X-register.

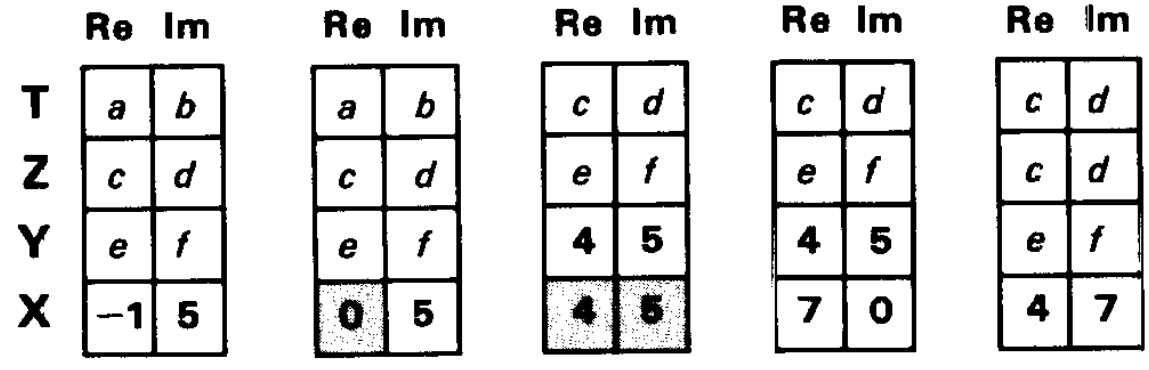
Example: Replace $-1-8i$ by $-1+5i$.



Keys: [Re≷Im] [←] 5 [Re≷Im] (continue with any operation)

Clearing the Real and Imaginary X-Registers. If you want to clear or replace *both* the real and imaginary parts of the number in the X-register, simply press [←], which will disable the stack, and enter your new number. (Enter zeros if you want the X-register to contain zeros.) Alternatively, if the new number will be purely real (including $0+0i$), you can quickly clear or replace the old, complex number by pressing [R↓] followed by zero or the new, real number.

Example: Replace $-1+5i$ with $4+7i$.

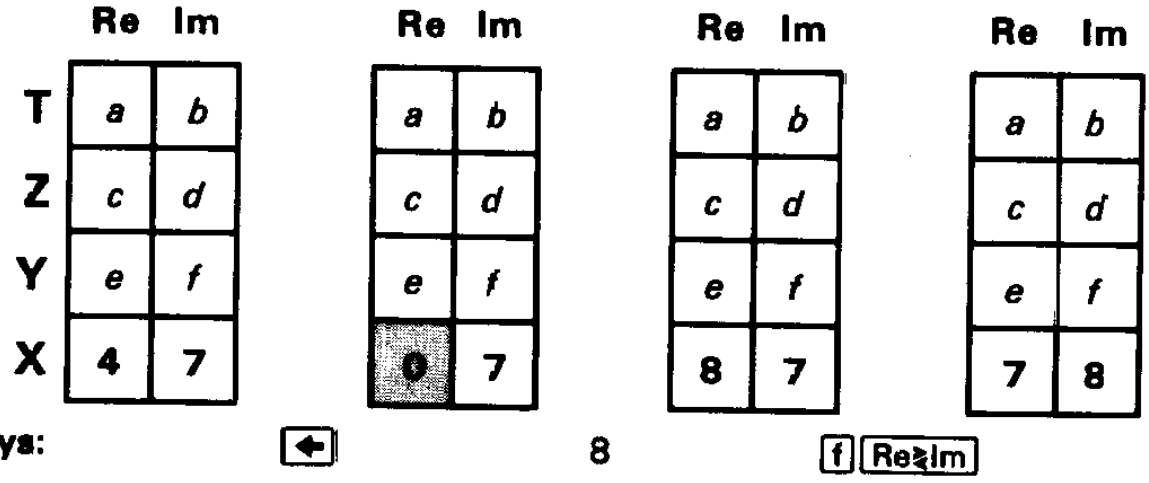


Keys: [←] 4 [ENTER] 7 [f] [I] (continue with any operation)

Entering Complex Numbers with [←]. The clearing functions [←] and [CLx] can also be used with [Re≷Im] as an alternative method of entering (and clearing) complex numbers. Using this method, you can enter a complex number using only the X-register, without affecting the rest of the stack. (This is possible because [←] and [CLx] disable stack lift.) Executing [Re≷Im] will also create an imaginary stack if one is not already present.

Example: Enter $9 + 8i$ without moving the stack and then find its square.

Keystrokes	Display	
([←])	(0.0000)	Prevents stack lift when the next digit (8) is keyed in. Omit this step if you'd rather save what's in X and lose what's in T.
8	8	Enter imaginary part first.
[f][Re≷Im]	7.0000	Displays real part; Complex mode activated.
[←]	0.0000	Disables stack. (Otherwise, it would lift following [Re≷Im].)
9	9	Enters real part (digit entry not terminated).
[g][x^2]	17.0000	Real part.
[f][i] (hold)	144.0000	Imaginary part.
(release)	17.0000	



	Re	Im
T	*a*	*b*
Z	*c*	*d*
Y	*e*	*f*
X	7	8

Keys: [←]

	Re	Im
	a	*b*
	c	*d*
	e	*f*
	0	8

9

	Re	Im
	a	*b*
	c	*d*
	e	*f*
	9	8

[g] [x^2]

	Re	Im
	a	*b*
	c	*d*
	e	*f*
	17	144

Entering a Real Number

You have already seen two ways of entering a complex number. There is a shorter way to enter a real number: simply key it (or recall it) into the display just as you would if the calculator were not in Complex mode. As you do so, a zero will be placed in the imaginary X-register (as long as the previous operation was not [←] or [CLx], as explained on page 124).

The operation of the real and imaginary stacks during this process is illustrated below. (Assume the last key pressed was not [←] or [CLx] and the contents remain from the previous example.)

	Re	Im
T	*a*	*b*
Z	*c*	*d*
Y	*e*	*f*
X	17	144

Keys: 4

	Re	Im
	c	*d*
	e	*f*
	17	144
	4	0

[ENTER]

	Re	Im
	e	*f*
	17	144
	4	0
	4	0

(Followed by another number.)

Entering a Pure Imaginary Number

There is a shortcut for entering a pure imaginary number into the X-register when you are already in Complex mode: key in the (imaginary) number and press [f] [Re≷Im].

Example: Enter $0 + 10i$ (assuming the last function executed was not [←] or [CLx]).

Keystrokes	Display	
10	**10**	Keys 10 into the displayed real X-register and zero into the imaginary X-register.
[f] [Re≷Im]	**0.0000**	Exchanges numbers in real and imaginary X-registers. Display again shows that the number in the real X-register is zero—as it should be for a pure imaginary number.

The operation of the real and imaginary stacks during this process is illustrated below. (Assume the stack registers contain the numbers resulting from the preceding examples.)

	Re	Im
T	*e*	*f*
Z	17	144
Y	4	0
X	4	0

Keys: 10

	Re	Im
	e	*f*
	17	144
	4	0
	10	0

[f] [Re≷Im]

Re	Im
e	*f*
17	144
4	0
0	10

(Continue with any operation.)

Note that pressing [f] [Re≷Im] simply exchanges the numbers in the real and imaginary *X-registers* and *not* those in the remaining stack registers.

Storing and Recalling Complex Numbers

The [STO] and [RCL] functions act on the *real X-register only*; therefore, the imaginary part of a complex number must be stored or recalled separately. The keystrokes to do this can be entered as part of a program and executed automatically.*

To store $a + ib$ from the complex X-register to R_1 and R_2, you can use the sequence

[STO] 1 [f] [Re≷Im] [STO] 2

You can follow this by [f] [Re≷Im] to return the stack to its original condition if desired. To recall $a + ib$ from R_1 and R_2 you can use the sequence

[RCL] 1 [RCL] 2 [f] [I]

If you wish to avoid disturbing the rest of the stack, you can recall the number using the sequence

[RCL] 2 [f] [Re≷Im] [←] [RCL] 1.

(In Program mode, use [g] [CLx] instead of [←].)

Operations With Complex Numbers

Almost all functions performed on *real numbers* will yield the same answer whether executed in or out of Complex mode,† *assuming the result is also real.* In other words, Complex mode does not restrict your ability to calculate with real numbers.

Any functions not mentioned below or in the rest of this section (Calculating With Complex Numbers) ignore the imaginary stack.

---

* You can use the HP-15C matrix functions, described in section 12, to make storing and recalling complex numbers more convenient. By dimensioning a matrix to be $n \times 2$, n complex numbers can be stored as rows of the matrix. (This technique is demonstrated in the *HP-15C Advanced Functions Handbook*, section 3, under Applications.)

† The exceptions are [→P] and [→R], which operate differently in Complex mode in order to facilitate converting complex numbers to polar form (page 133).

One-Number Functions

The following functions operate on both the real and imaginary parts of the number in the X-register, and place the real and imaginary parts of the answer back into those registers.

[√x] [x²] [LN] [LOG] [1/x] [10^x] [e^x] [ABS] [→P] [→R]

All trigonometric and hyperbolic functions and their inverses also belong to this group.*

The [ABS] function gives the magnitude of the number in the X-registers (the square root of the sum of the squares of the real and imaginary parts); the imaginary part of the magnitude is zero.

[→P] converts to polar form and [→R] converts to rectangular form, as described later in this section (page 133).

For the trigonometric functions, the calculator considers numbers in the real and imaginary X-registers to be expressed in *radians*—regardless of the current trigonometric mode. To calculate trigonometric functions for values given in degrees, use [→RAD] to convert those values to radians before executing the trigonometric function.

Two-Number Functions

The following functions operate on both the real and imaginary parts of the numbers in the X- and Y-registers, and place the real and imaginary parts of the answer into the X-registers. Both stacks drop, just as the ordinary stack drops after a two-number function *not* in Complex mode.

[+] [−] [×] [÷] [y^x]

Stack Manipulation Functions

When the calculator is in Complex mode, the following functions simultaneously manipulate both the real and imaginary stacks in the same way as they manipulate the ordinary stack when the calculator is not in Complex mode. The [x≷y] function, for instance,

---

*Refer to the *HP-15C Advanced Functions Handbook* for definitions of complex trigonometric functions and further information about doing calculations in Complex mode.

will exchange *both* the real and imaginary parts of the numbers in the X- and Y-registers.

[x≷y] [R↓] [R↑] [ENTER] [LSTx]

Conditional Tests

For programming, the four conditional tests below will work in the complex sense: [x=0] and [TEST] 0 compare the *complex* number in the (real and imaginary) X-registers to $0 + 0i$, while [TEST] 5 and [TEST] 6 compare the *complex* numbers in the (real and imaginary) X- and Y-registers. All other conditional tests besides those listed below ignore the imaginary stack.

[x=0] [TEST] 0 ($x \neq 0$) [TEST] 5 ($x = y$) [TEST] 6 ($x \neq y$)

Example: Complex Arithmetic. The characteristic impedance of a ladder network is given by an equation of the form

$$Z_0 = \sqrt{\frac{A}{B}},$$

where A and B are complex numbers. Find Z_0 for the hypothetical values $A = 1.2 + 4.7i$ and $B = 2.7 + 3.2i$.

Keystrokes	Display	
1.2 [ENTER] 4.7 [f] [I]	**1.2000**	Enters A into real and imaginary X-registers.
2.7 [ENTER] 3.2 [f] [I]	**2.7000**	Enters B into real and imaginary X-registers, moving A into real and imaginary Y-registers.
[÷]	**1.0428**	Calculates A/B.
[√x]	**1.0491**	Calculates Z_0 and displays real part.
[f] [(i)] (hold)	**0.2406**	Displays imaginary part of Z_0 while [(i)] is held down.
(release)	**1.0491**	Again displays real part of Z_0.

Complex Results from Real Numbers

In the preceding examples, the entry of complex numbers had ensured the (automatic) activation of Complex mode. There will be times, however, when you will need Complex mode to perform certain operations on *real* numbers, such as $\sqrt{-5}$. (Without Complex mode, such as operation would result in an **Error 0**—improper math function.) To activate Complex mode at any time *and without disturbing the stack contents*, set flag 8 before executing the function in question.*

Example: The arc sine ($\sin^{-1}$) of 2.404 normally would result in an **Error 0**. Assuming 2.404 in the X-register, the complex value arc sin 2.404 can be calculated as follows:

Keystrokes	Display	
[g] [SF] 8		Activates Complex Mode.
[g] [SIN^{-1}]	**1.5708**	Real part of arc sin 2.404.
[f] [(i)] (hold)	**-1.5239**	Imaginary part of arc sin 2.404.
(release)	**1.5708**	Display shows real part again when [(i)] is released.

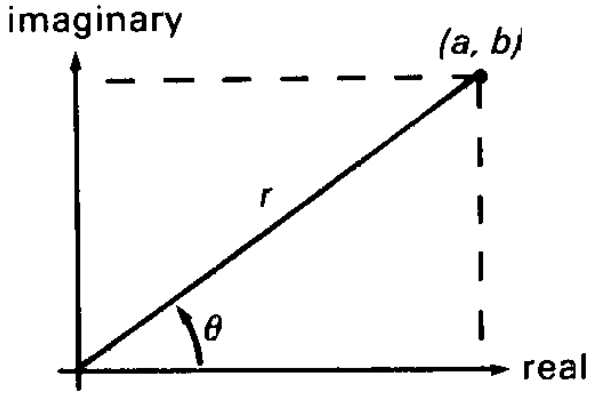
Polar and Rectangular Coordinate Conversions

In many applications, complex numbers are represented in polar form, sometimes using phasor notation. However, the HP-15C assumes that any complex numbers are in *rectangular* form. Therefore, any numbers in polar or phasor form must be converted to rectangular form before performing a function in Complex mode.

---

* Pressing [f] [Re≷Im] *twice* will accomplish the same thing. The sequence [f] [I] is not used because it would combine any numbers in the real X- and Y-registers into a single complex number.

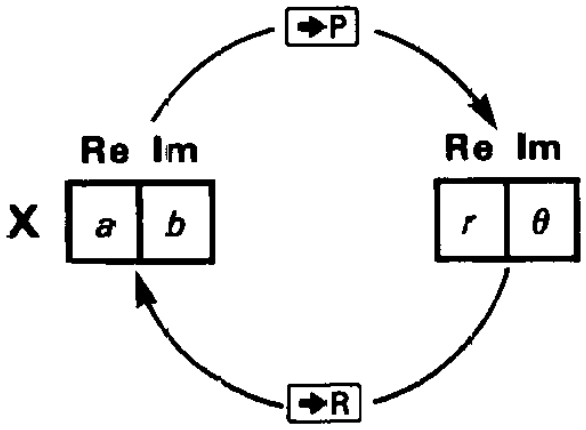
$$a + ib = \begin{cases} r(\cos\theta + i\sin\theta) = re^{i\theta} & \text{(polar)} \\ r\angle\theta & \text{(phasor)} \end{cases}$$



[→R] and [→P] can be used to interconvert the rectangular and polar forms of a complex number. They operate *in Complex mode* as follows:

[f] [→R] converts the polar (or phasor) form of a complex number to its rectangular form by replacing the magnitude r in the real X-register with a, and replacing the angle θ in the imaginary X-register with b.

[g] [→P] converts the rectangular coordinates of a complex number to the polar (or phasor) form by replacing the real part a in the real X-register with r, and replacing the imaginary part b in the imaginary X-register with θ.



These are the only functions in Complex mode that are affected by the current trigonometric mode setting. That is, the angular units for θ must correspond to the trigonometric mode indicated by the annunciator (or absence thereof).

Example: Find the sum $2(\cos 65° + i \sin 65°) + 3(\cos 40° + i \sin 40°)$ and express the result in polar form. (In phasor form, evaluate $2\angle 65° + 3\angle 40°$.)

Keystrokes	Display	
g DEG		Sets Degrees mode for any polar-rectangular conversions.
2 ENTER	2.0000	
65 f I	2.0000	C annunciator displayed; Complex mode activated.
f →R	0.8452	Converts polar to rectangular form; real part (a) displayed.
3 ENTER	3.0000	
40 f I	3.0000	
f →R	2.2981	Converts polar to rectangular form; real part (a) displayed.
+	3.1434	
g →P	4.8863	Converts rectangular to polar form; r displayed.
f (i) (hold)	49.9612	θ (in degrees).
(release)	4.8863	

Problems

By working through the following problems, you will see that calculating with complex numbers on the HP-15C is as easy as calculating with real numbers. In fact, once your numbers are entered, most mathematical operations will use exactly the same keystrokes. Try it and see!

1. Evaluate: $$\frac{2i(-8+6i)^3}{(4-2\sqrt{5}\,i)(2-4\sqrt{5}\,i)}$$

Keystrokes	Display	
2 [f] [Re≷Im]	0.0000	$2i$. Display shows real part.
8 [CHS] [ENTER]	−8.0000	
6 [f] [I]	−8.0000	$-8+6i$.
3 [y^x]	352.0000	$(-8+6i)^3$.
[×]	−1,872.0000	$2i\,(-8+6i)^3$.
4 [ENTER]	4.0000	
5 [$\sqrt{x}$]	2.2361	
2 [CHS] [×]	−4.4721	$-2\sqrt{5}$.
[f] [I]	4.0000	$4-2\sqrt{5}\,i$.
[÷]	−295.4551	$\dfrac{2i(-8+6i)^3}{4-2\sqrt{5}i}$.
2 [ENTER] 5 [$\sqrt{x}$]	2.2361	
4 [CHS] [×]	−8.9443	
[f] [I]	2.0000	$2-4\sqrt{5}\,i$.
[÷]	9.3982	Real part of result.
[f] [(i)]	−35.1344 9.3982	Answer: $9.3982-35.1344i$.

2. Write a program to evaluate the function $\omega = \dfrac{2z+1}{5z+3}$ for different values of z. (ω represents a linear fractional transformation, a class of conformal mappings.) Evaluate ω for $z = 1+2i$.

   (Answer: $0.3902 + 0.0122\,i$. One possible keystroke sequence is: [f] [LBL] [A] [ENTER] [ENTER] 2 [×] 1 [+] [$x\gtrless y$] 5 [×] 3 [+] [÷] [R/S] [f] [Re≷Im] [g] [RTN].)

3. Try your hand at a complex polynomial and rework the example on page 80. You can use the same program to evaluate $P(z) = 5z^4 + 2z^3$, where z is some complex number.

   Load the stack with $z = 7 + 0i$ and see if you get the same answer as before.

   (Answer: $12{,}691.0000 + 0.0000i$.)

   Now run the program for $z = 1 + i$.

   (Answer: $-24.0000 + 4.0000i$.)

For Further Information

The *HP-15C Advanced Functions Handbook* presents more detailed and technical aspects of using complex numbers in various functions with the HP-15C. Applications are included. The topics include:

- Accuracy considerations.
- Principal branches of multivalued functions.
- Complex contour integrals.
- Complex potentials.
- Storing and recalling complex numbers using a matrix.
- Calculating the *n*th roots of a complex number.
- Solving an equation for its complex roots.
- Using [SOLVE] and [∫] in Complex mode.

Section 12

Calculating With Matrices

The HP-15C enables you to perform matrix calculations, giving you the capability to handle advanced problems with ease. The calculator can work with up to five matrices, which are named **A** through **E** since they are accessed using the corresponding [A] through [E] keys. The HP-15C lets you specify the size of each matrix, store and recall the values of matrix elements, and perform matrix operations—for matrices with real or complex elements. (A summary of matrix functions is listed at the end of this section.)

A common application of matrix calculations is solving a system of linear equations. For example, consider the equations

$$3.8x_1 + 7.2x_2 = 16.5$$

$$1.3x_1 - 0.9x_2 = -22.1$$

for which you must determine the values of x_1 and x_2.

These equations can be expressed in matrix form as $\mathbf{AX} = \mathbf{B}$, where

$$\mathbf{A} = \begin{bmatrix} 3.8 & 7.2 \\ 1.3 & -0.9 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \text{and } \mathbf{B} = \begin{bmatrix} 16.5 \\ -22.1 \end{bmatrix}.$$

The following keystrokes show how easily you can solve this matrix problem using your HP-15C. (The matrix operations used in this example are explained in detail later in this section.)

First, dimension the two known matrices, **A** and **B**, and enter the values of their elements, from left to right along each row from the first row to the last. Also, designate matrix **C** as the matrix that you will use to store the result of your matrix calculation (**C** = **X**).

Keystrokes	Display	
g CF 8		Deactivates Complex mode.
2 ENTER f DIM A	2.0000	Dimensions matrix **A** to be 2×2.
f MATRIX 1	2.0000	Prepares for automatic entry of matrix elements in User mode.
f USER	2.0000	(Turns on the **USER** annunciator.)
3.8 STO A	A 1,1	Denotes matrix **A**, row 1, column 1. (A display like this appears momentarily as you enter each element and remains as long as you hold the letter key.)
	3.8000	Stores a_{11}.
7.2 STO A	7.2000	Stores a_{12}.
1.3 STO A	1.3000	Stores a_{21}.
.9 CHS STO A	-0.9000	Stores a_{22}.
2 ENTER 1 f DIM B	1.0000	Dimensions matrix **B** to be 2×1.
16.5 STO B	16.5000	Stores b_{11}.
22.1 CHS STO B	-22.1000	Stores b_{21}.
f RESULT C	-22.1000	Designates matrix **C** for storing the result.

Using matrix notation, the solution of the matrix equation $\mathbf{AX} = \mathbf{B}$ is

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$$

where $\mathbf{A}^{-1}$ is the inverse of matrix **A**. You can perform this operation by entering the "descriptors" for matrices **B** and **A** into the Y- and X-registers and then pressing ÷. (A descriptor shows

the name and dimensions of a matrix.) Note that if **A** and **B** were numbers, you could calculate the answer in a similar manner.

Keystrokes	Display			
RCL MATRIX B	b	2	1	Enters descriptor for **B**, the 2×1 constant matrix.
RCL MATRIX A	A	2	2	Enters descriptor for **A**, the 2×2 coefficient matrix, into the X-register, moving the descriptor for **B** into the Y-register.
÷	running			Temporary display while $\mathbf{A}^{-1}\mathbf{B}$ is being calculated and stored in matrix **C**.
	C	2	1	Descriptor for the result matrix, **C**, a 2×1 matrix.

Now recall the elements of matrix **C**—the solution to the matrix equation. (Also remove the calculator from User mode and clear all matrices.)

Keystrokes	Display	
RCL C	C 1,1	Denotes matrix **C**, row 1, column 1.
	−11.2887	Value of c_{11} (x_1).
RCL C	8.2496	Value of c_{21} (x_2).
f USER	8.2496	Deactivates User mode.
f MATRIX 0	8.2496	Clears all matrices.

The solution to the system of equations is $x_1 = -11.2887$ and $x_2 = 8.2496$.

> **Note**: The description of matrix calculations in this section presumes that you are already familiar with matrix theory and matrix algebra.

Matrix Dimensions

Up to 64 matrix elements can be stored in memory. You can use all 64 elements in one matrix or distribute them among up to five

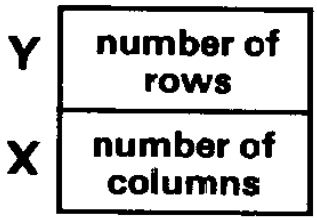
matrices. Matrix inversion, for example, can be performed on an 8×8 matrix with real elements (or on a 4×4 matrix with complex elements, as described later*).

To conserve memory, all matrices are initially dimensioned as 0×0. When a matrix is dimensioned or redimensioned, the proper number of registers is automatically allocated in memory. You may have to increase the number of registers allocated to matrix memory before dimensioning a matrix or before performing certain matrix operations. Appendix C describes how memory is organized, how to determine the number of registers currently available for storing matrix elements, and how to increase or decrease that number.

Dimensioning a Matrix

To dimension a matrix to have y rows and x columns, place those numbers in the Y- and X-registers, respectively, and then execute [f] [DIM] followed by the letter key specifying the matrix:

1. Key the number of rows (y) into the display, then press [ENTER] to lift it into the Y-register.
2. Key the number of columns (x) into the X-register.
3. Press [f] [DIM] followed by a letter key, [A] through [E], that specifies the name of the matrix.†



* The matrix functions described in this section operate on real matrices only. (In Complex mode, the imaginary stack is ignored during matrix operations.) However, the HP-15C has four matrix functions that enable you to calculate using real *representations* of complex matrices, as described on pages 160–173.

† You don't need to press [f] before the letter key. (Refer to Abbreviated Key Sequences on page 78.)

Example: Dimension matrix **A** to be a 2 × 3 matrix.

Keystrokes	Display	
2 [ENTER]	2.0000	Keys number of rows into Y-register.
3	3	Keys number of columns into X-register.
[f] [DIM] [A]	3.0000	Dimensions matrix **A** to be 2 × 3.

Displaying Matrix Dimensions

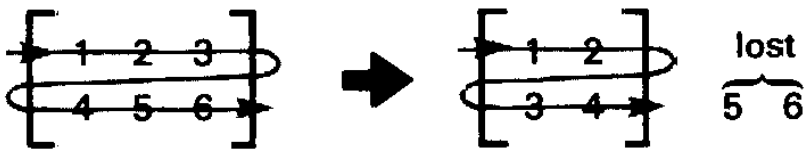
There are two ways you can display the dimensions of a matrix:

- Press [RCL] [MATRIX] followed by the letter key specifying the matrix. The calculator displays the name of the matrix at the left, and the number of rows followed by the number of columns at the right.
- Press [RCL] [DIM] followed by the letter key specifying the matrix. The calculator places the number of rows in the Y-register and the number of columns in the X-register.

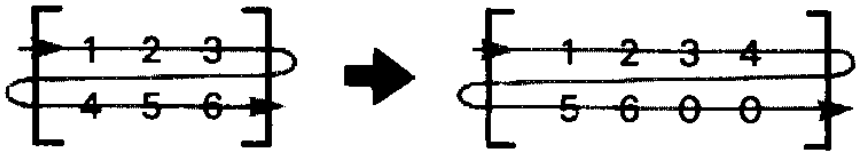
Keystrokes	Display	
[RCL] [MATRIX] [B]	b 0 0	Matrix B has 0 rows and 0 columns, since it has not been dimensioned otherwise.
[RCL] [DIM] [A]	3.0000	Number of columns in **A**.
[x⇄y]	2.0000	Number of rows in **A**.

Changing Matrix Dimensions

Values of matrix elements are stored in memory in order from left to right along each row, from the first row to the last. If you redimension a matrix to a smaller size, the required values are reassigned according to the new dimensions and the extra values are lost. For example, if the 2 × 3 matrix shown at the left below is redimensioned to 2 × 2, then



If you redimension a matrix to a larger size, elements with the value 0 are added at the end as required by the new dimensions. For example, if the same 2 × 3 matrix is redimensioned to 2 × 4, then



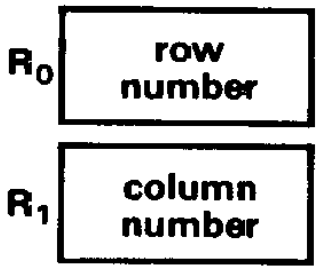
When you have finished calculating with matrices, you'll probably want to redimension all five matrices to 0 × 0, so that the registers used for storing their elements will be available for program lines or for other advanced functions. You can redimension all five matrices to 0 × 0 at one time by pressing f MATRIX 0. (You can dimension a single matrix to 0 × 0 by pressing 0 f DIM {A through E}.)

Storing and Recalling Matrix Elements

The HP-15C provides two ways of storing and recalling values of matrix elements. The first method allows you to progress through all of the elements in order. The second method allows you to access elements individually.

Storing and Recalling All Elements in Order

The HP-15C normally uses storage registers R_0 and R_1 to indicate the row and column numbers of a matrix element. If the calculator is in User mode, the row and column numbers are *automatically* incremented as you store or recall each matrix element, from left to right along each row from the first row to the last.



To set the row and column numbers in R_0 and R_1 to row 1, column 1, press f MATRIX 1.

To store or recall sequential elements of a matrix:

1. Be sure the matrix is properly dimensioned.
2. Press [f] [MATRIX] 1. This stores 1 in both storage registers R_0 and R_1, so that elements will be accessed starting at row 1, column 1.
3. Activate User mode by pressing [f] [USER]. With the calculator in User mode, after each element is stored or recalled the row number in R_0 or the column number in R_1 is automatically incremented by 1, as shown in the example following.
4. If you are storing elements, key in the value of the element to be stored in row 1, column 1.
5. Press [STO] or [RCL] followed by the letter key specifying the matrix.
6. Repeat steps 4 and 5 for all elements of the matrix. The row and column numbers are incremented according to the dimensions of the matrix you specify.

While the letter key specifying the matrix is held down after [STO] or [RCL] is pressed, the calculator displays the name of the matrix followed by the row and column numbers of the element whose value is being stored or recalled. If the letter key is held down for longer than about 3 seconds, the calculator displays **null**, doesn't store or recall the element value, and doesn't increment the row and column numbers. (Also, the stack registers aren't changed.)

After the last element of the matrix has been accessed, the row and column numbers both return to 1.

Example: Store the values shown below in the elements of the matrix **A** dimensioned above. (Be sure matrix **A** is dimensioned to 2×3.)

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Keystrokes	Display	
f MATRIX 1		Sets beginning row and column numbers in R_0 and R_1 to 1. (Display shows the previous result.)
f USER		Activates User mode.
1 STO A	A 1,1	Row 1, column 1 of **A**. (Displayed momentarily while A key held down.)
	1.0000	Value of a_{11}.
2 STO A	2.0000	Value of a_{12}.
3 STO A	3.0000	Value of a_{13}.
4 STO A	4.0000	Value of a_{21}.
5 STO A	5.0000	Value of a_{22}.
6 STO A	6.0000	Value of a_{23}.
RCL A	A 1,1	Recalls element in row 1, column 1. (R_0 and R_1 were reset in preceding step.)
	1.0000	Value of a_{11}.
RCL A	2.0000	Value of a_{12}.
RCL A	3.0000	Value of a_{13}.
RCL A	4.0000	Value of a_{21}.
RCL A	5.0000	Value of a_{22}.
RCL A	6.0000	Value of a_{23}.
f USER	6.0000	Deactivates User mode.

Checking and Changing Matrix Elements Individually

The calculator provides two ways to check (recall) and change (store) the value of a particular matrix element. The first method uses storage registers R_0 and R_1 in the same way as described above—except that the row and column numbers aren't automatically changed when User mode is deactivated. The second method uses the stack to define the row and column numbers.

Using R_0 and R_1. To access a particular matrix element, store its row number in R_0 and its column number in R_1. These numbers won't change automatically (unless the calculator is in User mode).

- To recall the element value (after storing the row and column numbers), press [RCL] followed by the letter key specifying the matrix.
- To store a value in that element (after storing the row and column numbers), place the value in the X-register and press [STO] followed by the letter key specifying the matrix.

Example: Store the value 9 as the element in row 2, column 3 of matrix **A** from the previous example.

Keystrokes	Display	
2 [STO] 0	**2.0000**	Stores row number in R_0.
3 [STO] 1	**3.0000**	Stores column number in R_1.
9	**9**	Keys the new element value into the X-register.
[STO] [A]	**A 2,3**	Row 2, column 3 of **A**.
	9.0000	Value of a_{23}.

Using the Stack. You can use the stack registers to specify a particular matrix element. This eliminates the need to change the numbers in R_0 and R_1.

- To recall an element value, enter the row number and column number into the stack (in that order). Then press [RCL] [g] followed by the letter key specifying the matrix. The element value is placed in the X-register. (The row and column numbers are lost from the stack.)

- To store an element value, first enter the value into the stack followed by the row number and column number. Then press [STO] [g] followed by the letter key specifying the matrix. (The row and column numbers are lost from the stack; the element value is returned to the X-register.)

Note that these are the only operations in which the blue [g] key precedes a gold letter key.

Example: Recall the element in row 2, column 1 of matrix **A** from the previous example. Use the stack registers.

Keystrokes	Display	
2 [ENTER] 1	1	Enters row number into Y-register and column number into X-register.
[RCL] [g] [A]	4.0000	Value of a_{21}.

Storing a Number in All Elements of a Matrix

To store a number in all elements of a matrix, simply key that number into the display, then press [STO] [MATRIX] followed by the letter key specifying the matrix.

Matrix Operations

In many ways, matrix operations are like numeric calculations. Numeric calculations require you to specify the numbers to be used; often you define a register for storing the result. Similarly, matrix calculations require you to specify one or two matrices that you want to use. A matrix *descriptor* is used to specify a particular matrix. For many calculations, you also must specify a matrix for storing the result. This is the *result matrix*.

Because matrix operations usually require many individual calculations, the calculator flashes the **running** display during most matrix operations.

Matrix Descriptors

Earlier in this section you saw that when you press [RCL] [MATRIX] followed by a letter key specifying a matrix, the name of the matrix appears at the left of the display and the number of rows followed by the number of columns appears at the right. The matrix name is called the *descriptor* of the matrix. Matrix descriptors can be moved among the stack and data storage registers just like a number—that is, using [STO], [RCL], [ENTER], etc. Whenever a matrix descriptor is displayed in the X-register, the *current* dimensions of that matrix are shown with it.

You use matrix descriptors to indicate which matrices are used in each matrix operation. The matrix operations discussed in the

rest of this section operate on the matrices whose descriptors are placed in the X-register and (for some operations) the Y-register.

Two matrix operations—calculating a determinant and solving the matrix equation **AX = B**—involve calculating an *LU* decomposition (also known as an *LU* factorization) of the matrix specified in the X-register.* A matrix that is an *LU* decomposition is signified by two dashes following the matrix name in the display of its descriptor. (Refer to page 160 for using a matrix in *LU* form.)

The Result Matrix

For many operations discussed in this section, you need to define the matrix in which the result of the operation should be stored. This matrix is called the *result matrix*.

Other matrix operations do *not* use or affect the result matrix. (This is noted in the descriptions of these operations.) Such an operation either replaces the original matrix with the result of the operation (if the result is a matrix, such as a transpose) or returns a number to the X-register (if the result is a number, such as a row norm).

Before you perform an operation that uses the result matrix, you must designate the result matrix. Do this by pressing [f] [RESULT] followed by the letter key specifying the matrix. (If the descriptor of the intended result matrix is already in the X-register, you can press [STO] [RESULT] instead.) The designated matrix remains the result matrix until another is designated.† To display the descriptor of the result matrix, press [RCL] [RESULT].

When you perform an operation that affects the result matrix, the matrix is automatically redimensioned to the proper size. If this redimensioning would require more additional elements than there are available in matrix memory (a *maximum* of 64 for all five matrices), then the operation can't be performed. This restriction

---

* The *LU* decomposition of a matrix **A** is another matrix in which is encoded a lower-triangular matrix, **L**, and an upper-triangular matrix, **U**, whose product **LU** equals matrix **A** (possibly with some rows interchanged). The *HP-15C Advanced Functions Handbook* discusses *LU* decomposition in detail.

† Matrix **A** is *automatically* designated as the result matrix whenever Continuous Memory is reset.

can often be overcome by designating the result matrix to be one of the matrices being operated on. (However, there are certain operations for which the result matrix can *not* be the same one as either of the matrices being operated on—this is noted in the description of these operations.)

While the key used for any matrix operation that stores a result in the result matrix is held down, the descriptor of the result matrix is displayed. If the key is released within about 3 seconds, the operation is performed, and the descriptor of the result matrix is placed in the X-register. If the key is held down longer, the operation is not performed and the calculator displays **null**.

Copying a Matrix

To copy the elements of a matrix into the corresponding elements of another matrix, use the [STO] [MATRIX] sequence:

1. Press [RCL] [MATRIX] followed by the letter key specifying the matrix to be copied. This enters the descriptor of the matrix into the display.
2. Press [STO] [MATRIX] followed by the letter key specifying the matrix to be copied into.

If the matrix specified after [RCL] does not have the same dimensions as the matrix specified after [STO], the second matrix is redimensioned to agree with the first. The matrix specified after [STO] need not already be dimensioned.

Example: Copy matrix **A** from the previous example into matrix **B**.

Keystrokes	Display			
[RCL] [MATRIX] [A]	**A**	**2**	**3**	Displays descriptor of matrix to be copied.
[STO] [MATRIX] [B]	**A**	**2**	**3**	Redimensions matrix **B** and copies **A** into **B**.
[RCL] [MATRIX] [B]	**b**	**2**	**3**	Displays descriptor of new matrix **B**.

One-Matrix Operations

The following table shows functions that operate on only the matrix specified in the X-register. Operations involving a single

matrix plus a number in another stack register are described under Scalar Operations (page 151).

Keystroke(s)	Result in X-register	Effect on Matrix Specified in X-register	Effect on Result Matrix
CHS	No change.	Changes sign of all elements.	None.‡
1/x	Descriptor of result matrix.	None.‡	Inverse of specified matrix. §
f MATRIX 4	Descriptor of transpose.	Replaced by transpose.	None.‡
f MATRIX 7	Row norm of specified matrix.*	None.	None.
f MATRIX 8	Frobenius or Euclidean norm of specified matrix.†	None.	None.
f MATRIX 9	Determinant of specified matrix.	None.‡	*LU* decomposition of specified matrix. §

* The row norm is the largest sum of the absolute values of the elements in each row of the specified matrix.

† The Frobenius or Euclidean norm is the square root of the sum of the squares of all elements in the specified matrix.

‡ Unless the result matrix is the same matrix specified in the X-register.

§ If the specified matrix is a *singular matrix* (that is, one that doesn't have an inverse), then the HP-15C modifies the *LU* form by an amount that is usually small compared to round-off error. For 1/x, the calculated inverse is the inverse of a matrix close to the original, singular matrix. (Refer to the *HP-15C Advanced Functions Handbook* for further information.)

Example: Calculate the transpose of matrix **B**. Matrix **B** was set in preceding examples to

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

Keystrokes	Display			
RCL MATRIX B	b	2	3	Displays descriptor of 2 × 3 matrix **B**.
f MATRIX 4	b	3	2	Descriptor of 3 × 2 transpose.

Matrix **B** (which you can view using RCL B in User mode) is now

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

Scalar Operations

Scalar operations perform arithmetic operations between a scalar (that is, a number) and each element of a matrix. The scalar and the descriptor of the matrix must be placed in the X- and Y-registers—in either order. (Note that the register position will affect the outcome of the − and ÷ functions.) The resulting values are stored in the corresponding elements of the result matrix.

The possible operations are shown in the following table.

Operation	Elements of Result Matrix*	
	Matrix in Y-Register Scalar in X-Register	**Scalar in Y-Register Matrix in X-Register**
[+]	Adds scalar value to each matrix element.	
[×]	Multiplies each matrix element by scalar value.	
[−]	Subtracts scalar value from each matrix element.	Subtracts each matrix element from scalar value.
[÷]	Divides each matrix element by scalar value.	Calculates inverse of matrix and multiplies each element by scalar value.

* Result matrix may be the specified matrix.

Example: Calculate the matrix $\mathbf{B} = 2\mathbf{A}$, then subtract 1 from every element in **B**. From before, use

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 9 \end{bmatrix}$$

Keystrokes	Display			
[f] [RESULT] [B]				Designates matrix **B** as result matrix.
[RCL] [MATRIX] [A]	**A**	**2**	**3**	Displays descriptor of matrix **A**.
2 [×]	**b**	**2**	**3**	Redimensions matrix **B** to the same dimensions as **A**, multiplies the elements of **A** by 2, stores those values in the corresponding elements of **B**, and displays the descriptor of the result matrix.

Keystrokes	Display			
1 [−]	b	2	3	Subtracts 1 from the elements of matrix **B** and stores those values in the same elements of **B**.

The result (which you can view using [RCL] [B] in User mode) is

$$\mathbf{B} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 17 \end{bmatrix}.$$

Arithmetic Operations

With matrix descriptors in both the X- and Y-registers, pressing [+] or [−] calculates the sum or difference of the matrices.

Pressing	Calculates*
[+]	**Y** + **X**
[−]	**Y** − **X**

*Result is stored in result matrix. Result matrix may be **X** or **Y**.

Example: Calculate **C** = **B** − **A**, where **A** and **B** are defined in the previous example,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 9 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 17 \end{bmatrix}.$$

Keystrokes	Display			
[f] [RESULT] [C]				Designates **C** as result matrix.
[RCL] [MATRIX] [B]	b	2	3	Recalls descriptor of matrix **B**. (This step can be skipped if descriptor is already in X-register.)
[RCL] [MATRIX] [A]	A	2	3	Recalls descriptor of matrix **A** into X-register, moving descriptor of matrix **B** to Y-register.

Keystrokes	Display	
[−]	C 2 3	Calculates **B** − **A** and stores values in redimensioned result matrix **C**.

The result is $\mathbf{C} = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 8 \end{bmatrix}$

Matrix Multiplication

With matrix descriptors in both the X- and Y-registers, you can calculate three different matrix products. The table below shows the results of the three functions for a matrix **X** specified in the X-register and a matrix **Y** specified in the Y-register. The matrix $\mathbf{X}^{-1}$ is the inverse of **X**, and the matrix $\mathbf{Y}^T$ is the transpose of **Y**.

Pressing	Calculates*
[×]	$\mathbf{Y}\,\mathbf{X}$
[f] [MATRIX] 5	$\mathbf{Y}^T\mathbf{X}$
[÷]	$\mathbf{X}^{-1}\mathbf{Y}$

*Result stored in result matrix. For [÷], the result matrix can be **Y** but not **X**. For the others, the result matrix must be other than **X** or **Y**.

Note: When you use the [÷] function to evaluate the expression $\mathbf{A}^{-1}\mathbf{B}$, you must enter the matrix descriptors in the order **B**, **A** rather than in the order that they appear in the expression.*

The value stored in each element of the result matrix is determined according to the usual rules of matrix multiplication.

For [MATRIX] 5, the matrix specified in the Y-register isn't changed by this operation, even though its transpose is used. The result is identical to that obtained using [MATRIX] 4 (transpose) and [×].

---

*This is the same order you would use if you were entering b and a for evaluating $a^{-1}b = b/a$.

For [÷], the matrix specified in the X-register is replaced by its LU decomposition. The [÷] function calculates $\mathbf{X}^{-1}\,\mathbf{Y}$ using a more direct method than does [1/x] and [×], giving the result faster and with improved accuracy.

Example: Using matrices **A** and **B** from the previous example, calculate $\mathbf{C} = \mathbf{A}^T\mathbf{B}$.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 9 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 17 \end{bmatrix}$$

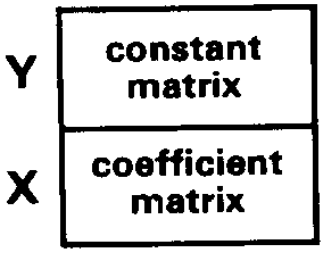
Keystrokes	Display			
[RCL] [MATRIX] [A]	A	2	3	Recalls descriptor for matrix **A**.
[RCL] [MATRIX] [B]	b	2	3	Recalls descriptor for matrix **B** into X-register, moving matrix **A** descriptor into Y-register.
[f] [RESULT] [C]	b	2	3	Designates matrix **C** as result matrix.
[f] [MATRIX] 5	C	3	3	Calculates $\mathbf{A}^T\mathbf{B}$ and stores result in matrix **C**, which is redimensioned to 3×3.

The result, matrix **C**, is

$$\mathbf{C} = \begin{bmatrix} 29 & 39 & 73 \\ 37 & 51 & 95 \\ 66 & 90 & 168 \end{bmatrix}.$$

Solving the Equation AX = B

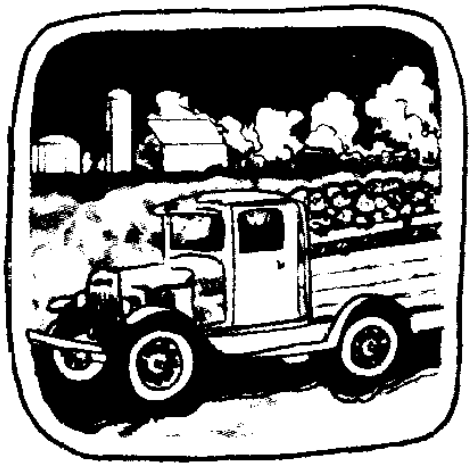
The [÷] function is useful for solving matrix equations of the form **AX** = **B**, where **A** is the coefficient matrix, **B** is the constant matrix, and **X** is the solution matrix. The descriptor of the constant matrix **B** should be entered in the Y-register and the descriptor of the coefficient matrix **A** should be entered in the X-register. Pressing [÷] then calculates the solution $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$.*



Remember that the [÷] function replaces the coefficient matrix by its *LU* decomposition and that this matrix must not be specified as the result matrix. Furthermore, using [÷] rather than [1/x] and [×] gives a solution faster and with improved accuracy.

At the beginning of this section, you found the solution for a system of linear equations in which the constant matrix and the solution matrix each had one column. The following example illustrates that you can use the HP-15C to find solutions for more than one set of constants—that is, for a constant matrix and solution matrix with more than one column.

Example: Looking at his receipts for his last three deliveries of cabbage and broccoli, Silas Farmer sees the following summary.



* If **A** is a *singular* matrix (that is, one that doesn't have an inverse), then the HP-15C modifies the *LU* form of **A** by an amount that is usually small compared to round-off error. The calculated solution corresponds to that for a nonsingular coefficient matrix close to the original, singular matrix.

	Week		
	1	2	3
Total Weight (kg)	274	233	331
Total Value	$120.32	$112.96	$151.36

Silas knows that he received $0.24 per kilogram for his cabbage and $0.86 per kilogram for his broccoli. Use matrix operations to determine the weights of cabbage and broccoli he delivered each week.

Solution: Each week's delivery represents two linear equations (one for weight and one for value) with two unknown variables (the weights of cabbage and broccoli). All three weeks can be handled simultaneously using the matrix equation

$$\begin{bmatrix} 1 & 1 \\ 0.24 & 0.86 \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \end{bmatrix} = \begin{bmatrix} 274 & 233 & 331 \\ 120.32 & 112.96 & 151.36 \end{bmatrix}$$

or

$$\mathbf{AD} = \mathbf{B}$$

where the first row of matrix **D** is the weights of cabbage for the three weeks and the second row is the weights of broccoli.

Keystrokes	Display	
2 [ENTER] [f] [DIM] [A]	2.0000	Dimensions **A** as 2×2 matrix.
[f] [MATRIX] 1	2.0000	Sets row and column numbers in R_0 and R_1 to 1.
[f] [USER]	2.0000	Activates User mode.
1 [STO] [A]	1.0000	Stores a_{11}.
[STO] [A]	1.0000	Stores a_{12}.
.24 [STO] [A]	0.2400	Stores a_{21}.
.86 [STO] [A]	0.8600	Stores a_{22}.
2 [ENTER] 3 [f] [DIM] [B]	3.0000	Dimensions **B** as 2×3 matrix.

Keystrokes	Display			
274 STO B	274.0000			Stores b_{11}.*
233 STO B	233.0000			Stores b_{12}.
331 STO B	331.0000			Stores b_{13}.
120.32 STO B	120.3200			Stores b_{21}.
112.96 STO B	112.9600			Stores b_{22}.
151.36 STO B	151.3600			Stores b_{23}.
f RESULT D	151.3600			Designates matrix **D** as result matrix.
RCL MATRIX B	b	2	3	Recalls descriptor of constant matrix.
RCL MATRIX A	A	2	2	Recalls descriptor of coefficient matrix **A** into X-register, moving descriptor of constant matrix **B** into Y-register.
÷	d	2	3	Calculates $\mathbf{A}^{-1}\mathbf{B}$ and stores result in matrix **D**.
RCL D	186.0000			Recalls d_{11}, the weight of cabbage for the first week.
RCL D	141.0000			Recalls d_{12}, the weight of cabbage for the second week.
RCL D	215.0000			Recalls d_{13}.
RCL D	88.0000			Recalls d_{21}.
RCL D	92.0000			Recalls d_{22}.
RCL D	116.0000			Recalls d_{23}.
f USER	116.0000			Deactivates User mode.

* Note that you did not need to press f MATRIX 1 before beginning to store the elements of matrix **B**. This is because after you stored the last element of matrix **A**, the row and column numbers in R_0 and R_1 were automatically reset to 1.

Silas' deliveries were:

	Week		
	1	2	3
Cabbage (kg)	186	141	215
Broccoli (kg)	88	92	116

Calculating the Residual

The HP-15C enables you to calculate the residual, that is, the matrix

$$\text{Residual} = \mathbf{R} - \mathbf{YX}$$

where **R** is the result matrix and **X** and **Y** are the matrices specified in the X- and Y-registers.

This capability is useful, for example, in doing iterative refinement on the solution of a system of equations and for linear regression problems. For example, if **C** is a possible solution for $\mathbf{AX} = \mathbf{B}$, then $\mathbf{B} - \mathbf{AC}$ indicates how well this solution satisfies the equation. (Refer to the *HP-15C Advanced Functions Handbook* for information about iterative refinement and linear regression.)

The residual function ([MATRIX] 6) uses the current contents of the result matrix and the matrices specified in the X- and Y-registers to calculate the residual defined above. The residual is stored in the result matrix, replacing the original result matrix. A matrix specified in the X- or Y-register can not be the result matrix.

Using [MATRIX] 6 rather than [×] and [−] gives a result with improved accuracy, particularly if the residual is small compared to the matrices being subtracted.

To calculate the residual:

1. Enter the descriptor of the **Y** matrix into the Y-register.
2. Enter the descriptor of the **X** matrix into the X-register.
3. Designate the **R** matrix as the result matrix.
4. Press [f] [MATRIX] 6. The residual replaces the original result matrix (**R**). The descriptor of the result matrix is placed in the X-register.

Using Matrices in *LU* Form

As noted earlier, two matrix operations (calculating a determinant and solving the matrix equation **AX** = **B**) create an *LU* decomposition of the matrix specified in the X-register. The descriptor of such a matrix has two dashes following the matrix name. A matrix in *LU* form has elements that differ from the elements of the original matrix.

However, the descriptor for a matrix in *LU* form can be used in place of the descriptor for the original matrix for operations involving the inverse of the matrix and for the determinant operation. That is, either the original matrix or its *LU* decomposition can be used for these operations:

[1/x]
[÷] for the matrix in the X-register
[MATRIX] 9

For these three functions, using the *LU* form of the matrix to be inverted gives a result that is identical to that using the original matrix.

As an example, if you solved the matrix equation **AX** = **B**, matrix **A** would be changed to its *LU* form. If you wanted to change the **B** matrix and solve the equation again, you could do so *without* changing the **A** matrix—the *LU* matrix will give the correct solution.

For all other matrix operations, a matrix that is an *LU* decomposition is *not* recognized as representing its original matrix. Instead, the elements of the *LU* matrix are used just as they appear in matrix memory and the result is not the result you would obtain using the original matrix.

Calculations With Complex Matrices

The HP-15C enables you to perform matrix multiplication and matrix inversion with complex matrices (that is, matrices whose elements are complex numbers) and to solve systems of complex equations (that is, equations whose coefficients and variables are complex).

However, the HP-15C stores and operates on only real matrices. The capability of doing calculations with complex matrices is

completely independent of the capability of doing calculations with complex numbers described in the preceding section. *You don't need to activate Complex mode for calculations with complex matrices.*

Instead, calculations with complex matrices are performed by using real matrices derived from the original complex matrices—in a manner to be described below—and performing certain transformations in addition to the regular matrix operations. These transformations are performed by four calculator functions. This section will describe how to do these calculations. (There are more examples of calculations with complex matrices in the *HP-15C Advanced Functions Handbook*.)

Storing the Elements of a Complex Matrix

Consider an $m \times n$ complex matrix $\mathbf{Z} = \mathbf{X} + i\mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are real $m \times n$ matrices. This matrix can be represented in the calculator as a $2m \times n$ "partitioned" matrix:

$$\mathbf{Z}^P = \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} \begin{matrix} \} \text{ Real Part} \\ \} \text{ Imaginary Part} \end{matrix}$$

The superscript P signifies that the complex matrix is represented by a partitioned matrix.

All of the elements of $\mathbf{Z}^P$ are real numbers—those in the upper half represent the elements of the real part (matrix $\mathbf{X}$), those in the lower half represent the elements of the imaginary part (matrix $\mathbf{Y}$). The elements of $\mathbf{Z}^P$ are stored in one of the five matrices (**A**, for example) in the usual manner, as described earlier in this section.

For example, if $\mathbf{Z} = \mathbf{X} + i\mathbf{Y}$, where

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix},$$

then **Z** can be represented in the calculator by

$$\mathbf{A}=\mathbf{Z}^P=\begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix}=\left[\begin{array}{cc}x_{11} & x_{12}\\ x_{21} & x_{22}\\ \hdashline y_{11} & y_{12}\\ y_{21} & y_{22}\end{array}\right].$$

Suppose you need to do a calculation with a complex matrix that is not written as the sum of a real matrix and an imaginary matrix—as was the matrix **Z** in the example above—but rather written with an entire complex number in each element, such as

$$\mathbf{Z}=\begin{bmatrix}x_{11}+iy_{11} & x_{12}+iy_{12}\\ x_{21}+iy_{21} & x_{22}+iy_{22}\end{bmatrix}.$$

This matrix can be represented in the calculator by a real matrix that looks very similar—one that is derived simply by ignoring the i and the + sign. The 2×2 matrix **Z** shown above, for example, can be represented in the calculator in "complex" form by the 2×4 matrix.

$$\mathbf{A}=\mathbf{Z}^C=\begin{bmatrix}x_{11} & y_{11} & x_{12} & y_{12}\\ x_{21} & y_{21} & x_{22} & y_{22}\end{bmatrix}.$$

The superscript C signifies that the complex matrix is represented in a "complex-like" form.

Although a complex matrix can be *initially* represented in the calculator by a matrix of the form shown for $\mathbf{Z}^C$, the transformations used for multiplying and inverting a complex matrix presume that the matrix is represented by a matrix of the form shown for $\mathbf{Z}^P$. The HP-15C provides two transformations that convert the representation of a complex matrix between $\mathbf{Z}^C$ and $\mathbf{Z}^P$:

Pressing	Transforms	Into
f Py,x	$\mathbf{Z}^C$	$\mathbf{Z}^P$
g Cy,x	$\mathbf{Z}^P$	$\mathbf{Z}^C$

To do either of these transformations, recall the descriptor of $\mathbf{Z}^C$ or $\mathbf{Z}^P$ into the display, then press the keys shown above. The

transformation is done to the specified matrix; the result matrix is not affected.

Example: Store the complex matrix

$$\mathbf{Z} = \begin{bmatrix} 4+3i & 7-2i \\ 1+5i & 3+8i \end{bmatrix}$$

in the form $\mathbf{Z}^C$, since it is written in a form that shows $\mathbf{Z}^C$. Then transform $\mathbf{Z}^C$ into the form $\mathbf{Z}^P$.

You can do this by storing the elements of $\mathbf{Z}^C$ in matrix **A** and then using the [Py,x] function, where

$$\mathbf{A} = \mathbf{Z}^C = \begin{bmatrix} 4 & 3 & 7 & -2 \\ 1 & 5 & 3 & 8 \end{bmatrix}.$$

Keystrokes	Display			
[f] [MATRIX] 0				Clears all matrices.
2 [ENTER] 4 [f] [DIM] [A]	4.0000			Dimensions matrix **A** to be 2×4.
[f] [MATRIX] 1	4.0000			Sets beginning row and column numbers in R_0 and R_1 to 1.
[f] [USER]	4.0000			Activates User mode.
4 [STO] [A]	4.0000			Stores a_{11}.
3 [STO] [A]	3.0000			Stores a_{12}.
7 [STO] [A]	7.0000			Stores a_{13}.
2 [CHS] [STO] [A]	-2.0000			Stores a_{14}.
1 [STO] [A]	1.0000			Stores a_{21}.
5 [STO] [A]	5.0000			Stores a_{22}.
3 [STO] [A]	3.0000			Stores a_{23}.
8 [STO] [A]	8.0000			Stores a_{24}.
[f] [USER]	8.0000			Deactivates User mode.
[RCL] [MATRIX] [A]	A	2	4	Displays descriptor of matrix **A**.
[f] [Py,x]	A	4	2	Transforms $\mathbf{Z}^C$ into $\mathbf{Z}^P$, and redimensions matrix **A**.

Matrix **A** now represents the complex matrix **Z** in $\mathbf{Z}^P$ form:

$$\mathbf{A} = \mathbf{Z}^P = \left[\begin{array}{cc} 4 & 7 \\ 1 & 3 \\ \hdashline 3 & -2 \\ 5 & 8 \end{array}\right] \begin{array}{l} \} \text{ Real Part} \\ \\ \} \text{ Imaginary Part} \end{array}$$

The Complex Transformations

An additional transformation must be done when you want to calculate the product of two complex matrices, and still another when you want to calculate the inverse of a complex matrix. These transformations convert between the $\mathbf{Z}^P$ representation of an $m \times n$ complex matrix and a $2m \times 2n$ partitioned matrix of the following form:

$$\tilde{\mathbf{Z}} = \begin{bmatrix} \mathbf{X} & -\mathbf{Y} \\ \mathbf{Y} & \mathbf{X} \end{bmatrix}.$$

The matrix $\tilde{\mathbf{Z}}$ created by the [MATRIX] 2 transformation has twice as many elements as $\mathbf{Z}^P$.

For example, the matrices below show how $\tilde{\mathbf{Z}}$ is related to $\mathbf{Z}^P$.

$$\mathbf{Z}^P = \left[\begin{array}{cc} 1 & -6 \\ \hdashline -4 & 5 \end{array}\right] \longleftrightarrow \tilde{\mathbf{Z}} = \left[\begin{array}{cc:cc} 1 & -6 & 4 & -5 \\ \hdashline -4 & 5 & 1 & -6 \end{array}\right]$$

The transformations that convert the represention of a complex matrix between $\mathbf{Z}^P$ and $\tilde{\mathbf{Z}}$ are shown in the following table.

Pressing	Transforms	Into
f [MATRIX] 2	$\mathbf{Z}^P$	$\tilde{\mathbf{Z}}$
f [MATRIX] 3	$\tilde{\mathbf{Z}}$	$\mathbf{Z}^P$

To do either of these transformations, recall the descriptor of $\mathbf{Z}^P$ or $\tilde{\mathbf{Z}}$ into the display, then press the keys shown above. The transformation is done to the specified matrix; the result matrix is not affected.

Inverting a Complex Matrix

You can calculate the inverse of a complex matrix by using the fact that $(\tilde{\mathbf{Z}})^{-1} = (\widetilde{\mathbf{Z}^{-1}})$.

To calculate the inverse, $\mathbf{Z}^{-1}$, of a complex matrix $\mathbf{Z}$:

1. Store the elements of $\mathbf{Z}$ in memory, in the form either of $\mathbf{Z}^P$ or of $\mathbf{Z}^C$.
2. Recall the descriptor of the matrix representing $\mathbf{Z}$ into the display.
3. If the elements of $\mathbf{Z}$ were entered in the form $\mathbf{Z}^C$, press [f] [Py,x] to transform $\mathbf{Z}^C$ into $\mathbf{Z}^P$.
4. Press [f] [MATRIX] 2 to transform $\mathbf{Z}^P$ into $\tilde{\mathbf{Z}}$.
5. Designate a matrix as the result matrix. It may be the same as the matrix in which $\tilde{\mathbf{Z}}$ is stored.
6. Press [1/x]. This calculates $(\tilde{\mathbf{Z}})^{-1}$, which is equal to $(\widetilde{\mathbf{Z}^{-1}})$. The values of these matrix elements are stored in the result matrix, and the descriptor of the result matrix is placed in the X-register.
7. Press [f] [MATRIX] 3 to transform $(\widetilde{\mathbf{Z}^{-1}})$ into $(\mathbf{Z}^{-1})^P$.
8. If you want the inverse in the form $(\mathbf{Z}^{-1})^C$, press [g] [Cy,x].

You can derive the complex elements of $\mathbf{Z}^{-1}$ by recalling the elements of $\mathbf{Z}^P$ or $\mathbf{Z}^C$ and then combining them as described earlier.

Example: Calculate the inverse of the complex matrix $\mathbf{Z}$ from the previous example,

$$\mathbf{A} = \mathbf{Z}^P = \left[\begin{array}{cc} 4 & 7 \\ 1 & 3 \\ \hdashline 3 & -2 \\ 5 & 8 \end{array}\right].$$

Keystrokes	Display			
[RCL] [MATRIX] [A]	A	4	2	Recalls descriptor of matrix A.
[f] [MATRIX] 2	A	4	4	Transforms $\mathbf{Z}^P$ into $\tilde{\mathbf{Z}}$, and redimensions matrix A.

Keystrokes	Display			
f RESULT B	A	4	4	Designates **B** as the result matrix.
1/x	b	4	4	Calculates $(\tilde{\mathbf{Z}})^{-1} = (\widetilde{\mathbf{Z}^{-1}})$ and places the result in matrix **B**.
f MATRIX 3	b	4	2	Transforms $(\widetilde{\mathbf{Z}^{-1}})$ into $(\mathbf{Z}^{-1})^P$.

The representation of $\mathbf{Z}^{-1}$ in partitioned form is contained in matrix **B**.

$$\mathbf{B} = \begin{bmatrix} -0.0254 & 0.2420 \\ -0.0122 & -0.1017 \\ \hline -0.2829 & -0.0022 \\ 0.1691 & -0.1315 \end{bmatrix} \begin{matrix} \} \text{ Real Part} \\ \\ \} \text{ Imaginary Part} \end{matrix}$$

Multiplying Complex Matrices

The product of two complex matrices can be calculated by using the fact that $(\mathbf{YX})^P = \tilde{\mathbf{Y}}\mathbf{X}^P$.

To calculate **YX**, where **Y** and **X** are complex matrices:

1. Store the elements of **Y** and **X** in memory, in the form either of $\mathbf{Z}^P$ or of $\mathbf{Z}^C$.
2. Recall the descriptor of the matrix representing **Y** into the display.
3. If the elements of **Y** were entered in the form $\mathbf{Y}^C$, press f Py,x to transform $\mathbf{Y}^C$ into $\mathbf{Y}^P$.
4. Press f MATRIX 2 to transform $\mathbf{Y}^P$ into $\tilde{\mathbf{Y}}$.
5. Recall the descriptor of the matrix representing **X** into the display.
6. If the elements of **X** were entered in the form $\mathbf{X}^C$, press f Py,x to transform $\mathbf{X}^C$ into $\mathbf{X}^P$.
7. Designate the result matrix; it must not be the same matrix as either of the other two.

8. Press [×] to calculate $\tilde{\mathbf{Y}}\mathbf{X}^P = (\mathbf{YX})^P$. The values of these matrix elements are placed in the result matrix, and the descriptor of the result matrix is placed in the X-register.
9. If you want the product in the form $(\mathbf{YX})^C$, press [g] [Cy,x].

Note that you don't transform $\mathbf{X}^P$ into $\tilde{\mathbf{X}}$.

You can derive the complex elements of the matrix product **YX** by recalling the elements of $(\mathbf{YX})^P$ or $(\mathbf{YX})^C$ and combining them according to the conventions described earlier.

Example: Calculate the product $\mathbf{ZZ}^{-1}$, where **Z** is the complex matrix given in the preceding example.

Since elements representing both matrices are already stored ($\tilde{\mathbf{Z}}$ in **A** and $(\mathbf{Z}^{-1})^P$ in **B**), skip steps 1, 3, 4, and 6.

Keystrokes	Display			
[RCL] [MATRIX] [A]	A	4	4	Displays descriptor of matrix **A**.
[RCL] [MATRIX] [B]	b	4	2	Displays descriptor of matrix **B**.
[f] [RESULT] [C]	b	4	2	Designates **C** as result matrix.
[×]	C	4	2	Calculates $\tilde{\mathbf{Z}}(\mathbf{Z}^{-1})^P = (\mathbf{ZZ}^{-1})^P$.
[f] [USER]	C	4	2	Activates User mode.
[RCL] [C]	C 1,1			Matrix **C**, row 1, column 1. (Displayed momentarily while last key held down.)
	1.0000			Value of c_{11}.
[RCL] [C]	−2.8500	−10		Value of c_{12}.
[RCL] [C]	−4.0000	−11		Value of c_{21}.
[RCL] [C]	1.0000			Value of c_{22}.
[RCL] [C]	1.0000	−11		Value of c_{31}.
[RCL] [C]	3.8000	−10		Value of c_{32}.
[RCL] [C]	1.0000	−11		Value of c_{41}.
[RCL] [C]	−1.0500	−10		Value of c_{42}.
[f] [USER]	−1.0500	−10		Deactivates User mode.

Writing down the elements of **C**,

$$\mathbf{C} = \left[\begin{array}{ll} 1.0000 & -2.8500\times10^{-10} \\ -4.0000\times10^{-11} & 1.0000 \\ \hline 1.0000\times10^{-11} & 3.8000\times10^{-10} \\ 1.0000\times10^{-11} & -1.0500\times10^{-10} \end{array}\right] = (\mathbf{ZZ}^{-1})^P,$$

where the upper half of matrix **C** is the real part of $\mathbf{ZZ}^{-1}$ and the lower half is the imaginary part. Therefore, by inspection of matrix **C**,

$$\mathbf{ZZ}^{-1} = \begin{bmatrix} 1.0000 & -2.8500\times10^{-10} \\ -4.0000\times10^{-11} & 1.0000 \end{bmatrix}$$

$$+i \begin{bmatrix} 1.0000\times10^{-11} & 3.8000\times10^{-11} \\ 1.0000\times10^{-11} & -1.0500\times10^{-10} \end{bmatrix}$$

As expected,

$$\mathbf{ZZ}^{-1} \cong \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + i\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Solving the Complex Equation AX = B

You can solve the complex matrix equation **AX** = **B** by finding $\mathbf{X} = \mathbf{A}^{-1}\,\mathbf{B}$. Do this by calculating $\mathbf{X}^P = (\tilde{\mathbf{A}})^{-1}\,\mathbf{B}^P$.

To solve the equation **AX** = **B**, where **A**, **X**, and **B** are complex matrices:

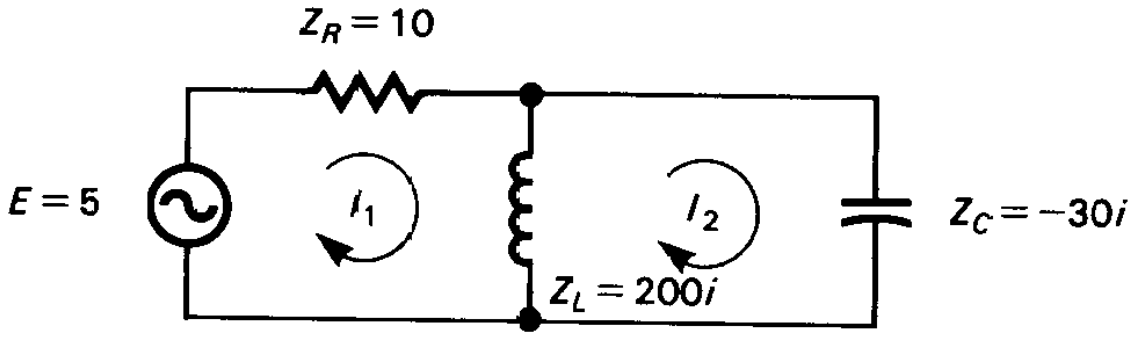
1. Store the elements of **A** and **B** in memory, in the form either of $\mathbf{Z}^P$ or of $\mathbf{Z}^C$.
2. Recall the descriptor of the matrix representing **B** into the display.
3. If the elements of **B** were entered in the form $\mathbf{B}^C$, press [f] [Py,x] to transform $\mathbf{B}^C$ into $\mathbf{B}^P$.

4. Recall the descriptor of the matrix representing **A** into the display.
5. If the elements of **A** were entered in the form of $\mathbf{A}^C$, press [f] [Py,x] to transform $\mathbf{A}^C$ into $\mathbf{A}^P$.
6. Press [f] [MATRIX] 2 to transform $\mathbf{A}^P$ into $\tilde{\mathbf{A}}$.
7. Designate the result matrix; it must not be the same as the matrix representing **A**.
8. Press [÷]; this calculates $\mathbf{X}^P$. The values of these matrix elements are placed in the result matrix, and the descriptor of the result matrix is placed in the X-register.
9. If you want the solution in the form $\mathbf{X}^C$, press [g] [Cy,x].

Note that you don't transform $\mathbf{B}^P$ into $\tilde{\mathbf{B}}$.

You can derive the complex elements of the solution **X** by recalling the elements of $\mathbf{X}^P$ or $\mathbf{X}^C$ and combining them according to the conventions described earlier.

Example: Engineering student A. C. Dimmer wants to analyze the electrical circuit shown below. The impedances of the components are indicated in complex form. Determine the complex representation of the currents I_1 and I_2.



This system can be represented by the complex matrix equation

$$\begin{bmatrix} 10+200i & -200i \\ -200i & (200-30)i \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

or

$$\mathbf{A}\,\mathbf{X} = \mathbf{B}.$$

In partitioned form,

$$\mathbf{A} = \left[\begin{array}{cc} 10 & 0 \\ 0 & 0 \\ \hline 200 & -200 \\ -200 & 170 \end{array}\right] \text{ and } \mathbf{B} = \left[\begin{array}{c} 5 \\ 0 \\ \hline 0 \\ 0 \end{array}\right],$$

where the zero elements correspond to real and imaginary parts with zero value.

Keystrokes	Display			Comments
4 [ENTER] 2 [f] [DIM] [A]	**2.0000**			Dimensions matrix **A** to be 4×2.
[f] [MATRIX] 1	**2.0000**			Set beginning row and column numbers in R_0 and R_1 to 1.
[f] [USER]	**2.0000**			Activates User mode.
10 [STO] [A]	**10.0000**			Stores a_{11}.
0 [STO] [A]	**0.0000**			Stores a_{12}.
[STO] [A]	**0.0000**			Stores a_{21}.
[STO] [A]	**0.0000**			Stores a_{22}.
200 [STO] [A]	**200.0000**			Stores a_{31}.
[CHS] [STO] [A]	**-200.0000**			Stores a_{32}.
[STO] [A]	**-200.0000**			Stores a_{41}.
170 [STO] [A]	**170.0000**			Stores a_{42}.
4 [ENTER] 1 [f] [DIM] [B]	**1.0000**			Dimensions matrix **B** to be 4×1.
0 [STO] [MATRIX] [B]	**0.0000**			Stores value 0 in all elements of **B**.
5 [ENTER] 1 [ENTER]	**1.0000**			Specifies value 5 for row 1, column 1.
[STO] [g] [B]	**5.0000**			Stores value 5 in b_{11}.
[RCL] [MATRIX] [B]	**b**	**4**	**1**	Recalls descriptor for matrix **B**.
[RCL] [MATRIX] [A]	**A**	**4**	**2**	Places descriptor for matrix **A** into X-register, moving descriptor for matrix **B** into Y-register.

Keystrokes	Display			
f MATRIX 2	A	4	4	Transforms $\mathbf{A}^P$ into $\tilde{\mathbf{A}}$.
f RESULT C	A	4	4	Designates matrix **C** as result matrix.
÷	C	4	1	Calculates $\mathbf{X}^P$ and stores in **C**.
g $C_{y,x}$	C	2	2	Transforms $\mathbf{X}^P$ into $\mathbf{X}^C$.
RCL C	0.0372			Recalls c_{11}.
RCL C	0.1311			Recalls c_{12}.
RCL C	0.0437			Recalls c_{21}.
RCL C	0.1543			Recalls c_{22}.
f USER	0.1543			Deactivates User mode.
f MATRIX 0	0.1543			Redimensions all matrices to 0×0.

The currents, represented by the complex matrix **X**, can be derived from **C**:

$$\mathbf{X} = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0.0372 + 0.1311i \\ 0.0437 + 0.1543i \end{bmatrix}.$$

Solving the matrix equation in the preceding example required 24 registers of matrix memory—16 for the 4×4 matrix **A** (which was originally entered as a 4×2 matrix representing a 2×2 complex matrix), and four each for the matrices **B** and **C** (each representing a 2×1 complex matrix). (However, you would have used four fewer registers if the result matrix were matrix **B**.) Note that since **X** and **B** are not restricted to be vectors (that is, single-column matrices), **X** and **B** could have required more memory.

The HP-15C contains sufficient memory to solve, using the method described above, the complex matrix equation $\mathbf{AX} = \mathbf{B}$ with **X** and **B** having up to six columns if **A** is 2×2, or up to two columns if **A** is 3×3.* (The allowable number of columns doubles if the constant matrix **B** is used as the result matrix.) If **X** and **B** have more columns, or if **A** is 4×4, you can solve the equation using the

* If all available memory space is dimensioned to the common pool (MEM: 1 64 0-0). Refer to appendix C, Memory Allocation.

alternate method below. This method differs from the preceding one in that it involves separate inversion and multiplication operations and fewer registers.

1. Store the elements of **A** in memory, in the form either of $\mathbf{A}^P$ or of $\mathbf{A}^C$.
2. Recall the descriptor of the matrix representing **A** into the display.
3. If the elements of **A** were entered in the form $\mathbf{A}^C$, press [f] [Py,x] to transform $\mathbf{A}^C$ into $\mathbf{A}^P$.
4. Press [f] [MATRIX] 2 to transform $\mathbf{A}^P$ into $\tilde{\mathbf{A}}$.
5. Press [STO] [RESULT] to designate the matrix representing **A** as the result matrix.
6. Press [1/x] to calculate $(\tilde{\mathbf{A}})^{-1}$.
7. Redimension **A** to have half the number of rows as indicated in the display of its descriptor after the preceding step.
8. Store the elements of **B** in memory, in the form either of $\mathbf{B}^P$ or of $\mathbf{B}^C$.
9. Recall the descriptor of the matrix representing **A** into the display.
10. Recall the descriptor of the matrix representing **B** into the display.
11. If the elements of **B** were entered in the form $\mathbf{B}^C$, press [f] [Py,x] to transform $\mathbf{B}^C$ into $\mathbf{B}^P$.
12. Press [f] [MATRIX] 2 to transform $\mathbf{B}^P$ into $\tilde{\mathbf{B}}$.
13. Designate the result matrix; it must not be the same matrix as either of the other two.
14. Press [×].
15. Press [f] [MATRIX] 4 to transpose the result matrix.
16. Press [f] [MATRIX] 2.
17. Redimension the result matrix to have half the number of rows as indicated in the display of its descriptor after the preceding step.

18. Press [RCL] [RESULT] to recall the descriptor of the result matrix.
19. Press [f] [MATRIX] 4 to calculate $\mathbf{X}^P$.
20. If you want the solution in the form $\mathbf{X}^C$, press [g] [Cy,x].

A problem using this procedure is given in the *HP-15C Advanced Functions Handbook* under Solving a Large System of Complex Equations.

Miscellaneous Operations Involving Matrices

Using a Matrix Element With Register Operations

If a letter key specifying a matrix is pressed after any of the following function keys, the operation is performed using the matrix element specified by the row and column numbers in R_0 and R_1, just as though it were a data storage register.

[STO] *
[STO] {[+], [−], [×], [÷]}
[DSE]
[x≷]

[RCL] *
[RCL] {[+], [−], [×], [÷]}
[ISG]

Using Matrix Descriptors in the Index Register

In certain applications, you may want to perform a programmed sequence of matrix operations using any of the matrices **A** through **E**. In this situation, the matrix operations can refer to whatever matrix descriptor is stored in the index register (R_I).

If the index register contains a matrix descriptor:

- Pressing [(i)] after any of the functions listed above performs the operation using the element specified by R_0 and R_1 and the matrix specified in R_I.
- Pressing [(i)] after [STO] [g] or [RCL] [g] performs the operation using the element specified by the row and column numbers in the Y- and X-registers and the matrix specified in R_I.

---

* Also, in User mode the row and column numbers in R_0 and R_1 are incremented according to the dimensions of the specified matrix.

- Pressing [f] [DIM] [I] dimensions the matrix specified in R_I according to the dimensions in the X- and Y-registers.
- Pressing [RCL] [DIM] [I] recalls to the X- and Y-registers the dimensions of the matrix specified in R_I.
- Pressing [GSB] [I] or [GTO] [I] has the same result as pressing [GSB] or [GTO] followed by the letter of the matrix specified in R_I. (This is not actually a matrix operation—only the letter in the matrix descriptor is used.)

Conditional Tests on Matrix Descriptors

Four conditional tests—[x=0], [TEST] 0 ($x \neq 0$), [TEST] 5 ($x = y$), and [TEST] 6 ($x \neq y$)—can be performed with matrix descriptors in the X- and Y-registers. Conditional tests can be used to control program execution, as described in section 8.

If a matrix descriptor is in the X-register, the result of [x=0] will be false and the result of [TEST] 0 will be true (regardless of the element values in the matrix.)

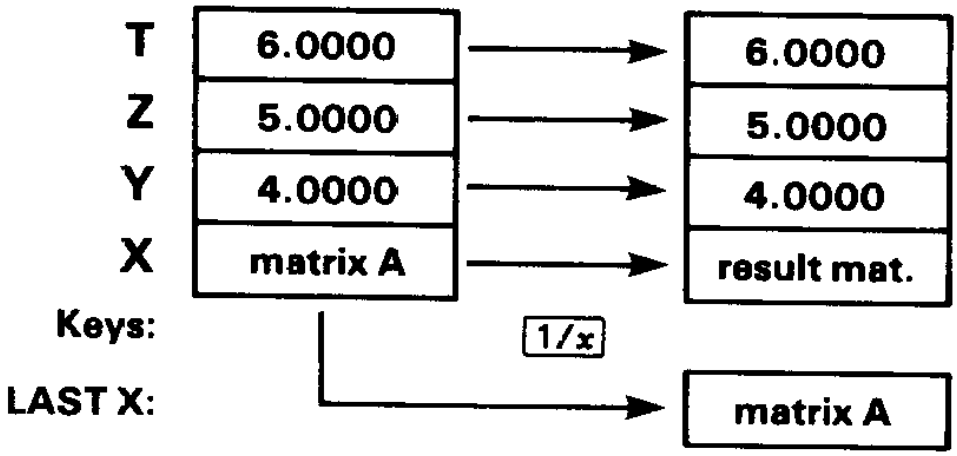
If matrix descriptors are in the X- and Y-registers when [TEST] 5 or [TEST] 6 conditional test is performed, x and y are equal if the same descriptor is in the X- and Y-registers, and not equal otherwise. The comparison is made *between the descriptors themselves, not between the elements* of the specified matrices.

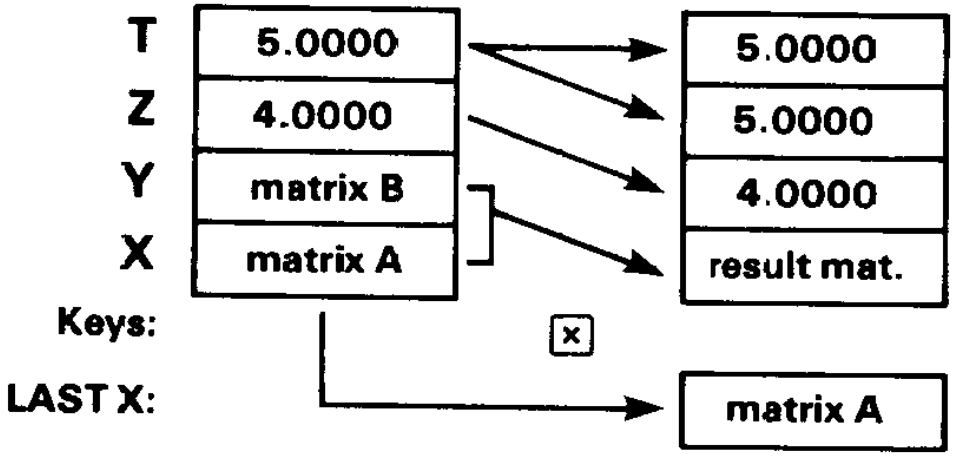
Other conditional tests can't be used with matrix descriptors.

Stack Operation for Matrix Calculations

During matrix calculations, the contents of the stack registers shift much like they do during numeric calculations.

For some matrix calculations, the result is stored in the result matrix. The arguments—one or two descriptors or numbers in the X-register or the X- and Y-registers—are combined by the operation, and the descriptor of the result matrix is placed in the X-register. (The argument from the X-register is placed in the LAST X register.)



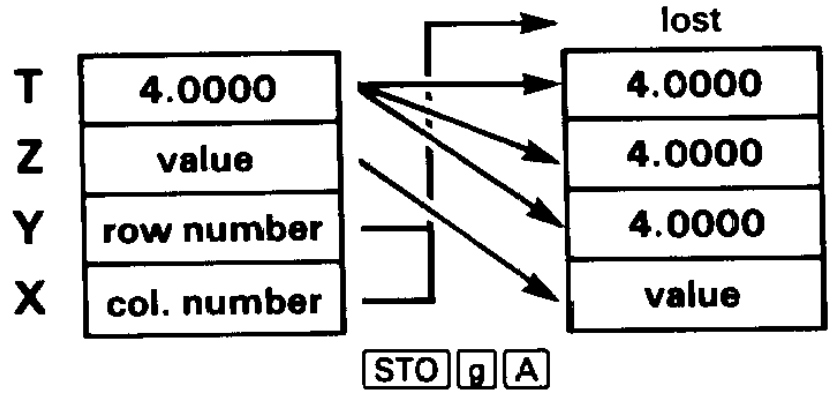


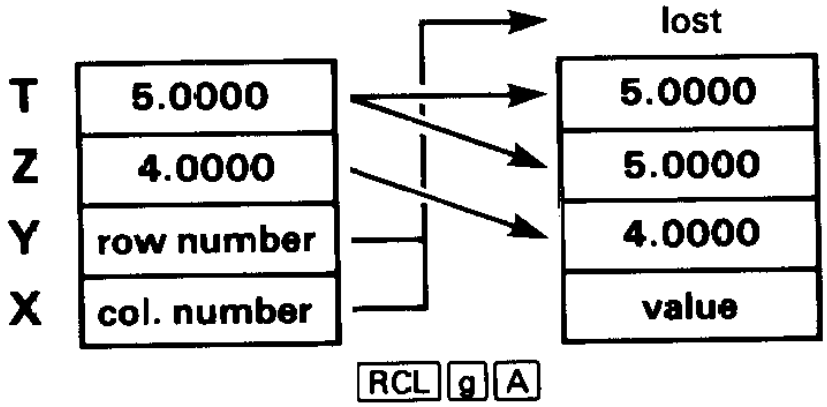
Several matrix functions operate on the matrix specified in the X-register only and store the result in the same matrix. For these operations the contents of the stack (including the LAST X register) are not moved—although the display changes to show the new dimensions if necessary.

For the [MATRIX] 7, [MATRIX] 8, and [MATRIX] 9 functions, the matrix descriptor specified in the X-register is placed in the LAST X register and the norm or (for [MATRIX] 9) the determinant is placed in the X-register. The Y-, Z-, and T-registers aren't changed.

When you recall descriptors or matrix elements into the X-register (with the stack enabled), other descriptors and numbers already in the stack move up in the stack—and the contents of the T-register are lost. (The LAST X register is not changed.) When you store descriptors or matrix elements, the stack (and the LAST X register) isn't changed.

In contrast to the operation described above, the [STO] [g] and [RCL] [g] functions do not affect the LAST X register and operate as shown on the next page.

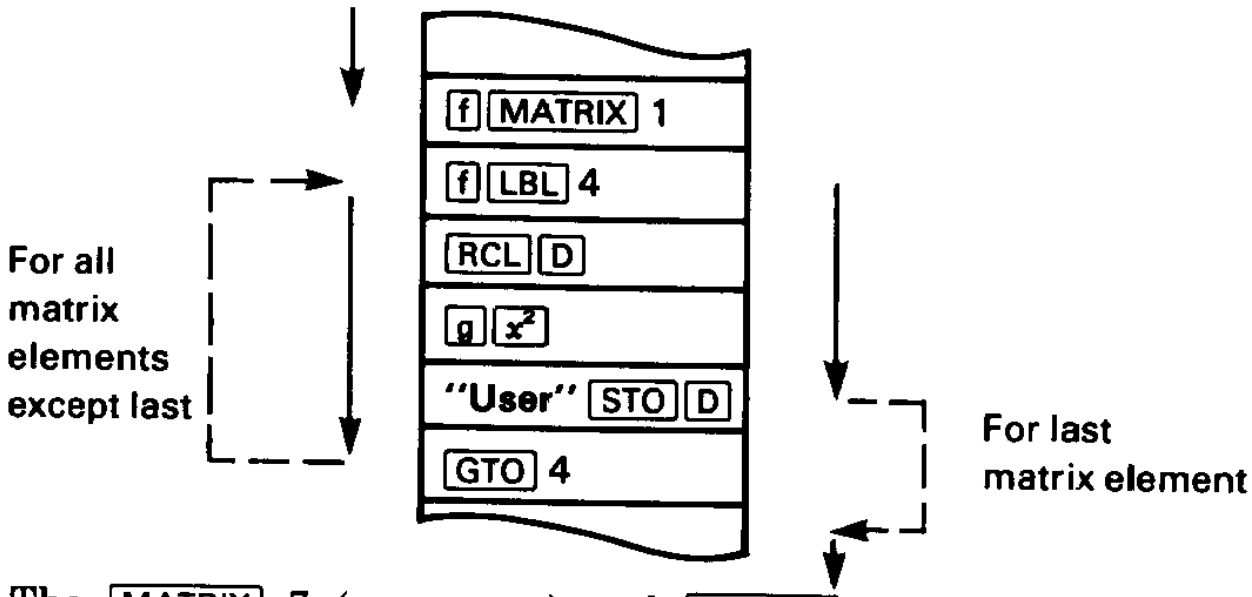




Using Matrix Operations in a Program

If the calculator is in User mode during program entry when you enter a [STO] or [RCL] {[A] through [E], [(i)]} instruction to store or recall a matrix element, a **u** replaces the dash usually displayed after the line number. When this line is executed in a running program, it operates as though the calculator were in User mode. That is, the row and column numbers in R_0 and R_1 are automatically incremented according to the dimensions of the specified matrix. This allows you to access elements sequentially. (The **USER** annunciator has no effect during program execution.)

In addition, when the last element is accessed by the "User" [STO] or [RCL] instruction—when R_0 and R_1 are returned to 1—program execution skips the next line. This is useful for programming a loop that stores or recalls each matrix element, then continues executing the program. For example, the following sequence squares all elements of matrix D:



The MATRIX 7 (row norm) and MATRIX 8 (Frobenius norm) functions also operate as conditional branching instructions in a program. If the X-register contains a matrix descriptor, these functions calculate the norm in the usual manner, and program execution continues with the next program line. If the X-register contains a number, program execution skips the next line. In both cases, the original contents of the X-register are stored in the LAST X register. This is useful for testing whether a matrix descriptor is in the X-register during a program.

Summary of Matrix Functions

Keystroke(s)	Results
g Cy,x	Transforms $\mathbf{Z}^P$ into $\mathbf{Z}^C$.
CHS	Changes sign of all elements in matrix specified in X-register.
f DIM {A through E, I}	Dimensions specified matrix.
f MATRIX 0	Dimensions all matrices to 0×0.
f MATRIX 1	Sets row and column numbers in R_0 and R_1 to 1.
f MATRIX 2	Transforms $\mathbf{Z}^P$ into $\tilde{\mathbf{Z}}$.
f MATRIX 3	Transforms $\tilde{\mathbf{Z}}$ into $\mathbf{Z}^P$.
f MATRIX 4	Calculates transpose of matrix specified in X-register.
f MATRIX 5	Multiplies transpose of matrix specified in Y-register with matrix specified in X-register. Stores in result matrix.

Keystroke(s)	Results
f MATRIX 6	Calculates residual in result matrix.
f MATRIX 7	Calculates row norm of matrix specified in X-register.
f MATRIX 8	Calculates Frobenius or Euclidean norm of matrix specified in X-register.
f MATRIX 9	Calculates determinant of matrix specified in X-register. Places *LU* in result matrix.
f $P_{y,x}$	Transforms $\mathbf{Z}^C$ into $\mathbf{Z}^P$
RCL { A through E, (i) }	Recalls value from specified matrix, using row and column numbers in R_0 and R_1.
RCL g { A through E, (i) }	Recalls value from specified matrix, using row and column numbers in Y- and X-registers.
RCL DIM { A through E, I }	Recalls dimensions of specified matrix into X- and Y-registers.
RCL MATRIX { A through E }	Displays descriptor of specified matrix.
RCL RESULT	Displays descriptor of result matrix.
f RESULT { A through E }	Designates specified matrix as result matrix.
STO { A through E, (i) }	Stores value from display into element of specified matrix, using row and column numbers in R_0 and R_1.
STO g { A through E, (i) }	Stores value from Z-register into element of specified matrix, using row and column numbers in Y- and X-registers.
STO MATRIX { A through E }	If matrix descriptor is in display, copies all elements of that matrix into corresponding elements of specified matrix. If number is in display, stores that value in all elements of specified matrix.

Keystroke(s)	Results
[STO] [RESULT]	Designates matrix specified in X-register as result matrix.
[f] [USER]	Row and column numbers in R_0 and R_1 are automatically incremented each time [STO] or [RCL] {[A] through [E], [(i)]} is pressed.
[1/x]	Inverts matrix specified in X-register. Stores in result matrix.
[+], [−]	If matrix descriptors specified in both X- and Y-registers, adds or subtracts corresponding elements of matrices specified. If matrix descriptor specified in only one of these registers, performs addition or subtraction with all elements in specified matrix and scalar in other register. Stores in result matrix.
[×]	If matrix descriptors specified in both X- and Y-registers, calculates product of specified matrices (as **YX**). If matrix specified in only one of these registers, multiplies all elements in specified matrix by scalar in other register. Stores in result matrix.
[÷]	If matrix descriptors specified in both X- and Y-registers, multiplies inverse of matrix specified in X-register with matrix specified in Y-register. If matrix specified in only Y-register, divides all elements of specified matrix by scalar in other register. If matrix specified in only X-register, multiplies each element of inverse of specified matrix by scalar in other register. Stores in result matrix.

For Further Information

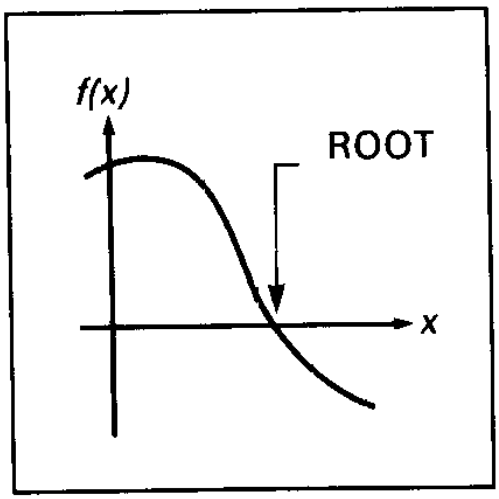
The *HP-15C Advanced Functions Handbook* presents more detailed and technical aspects of the matrix functions in the HP-15C, including applications. The topics include: least-squares calculations, solving nonlinear equations, ill-conditioned and singular matrices, accuracy considerations, iterative refinement, and creating the identity matrix.

Section 13

Finding the Roots of an Equation

In many applications you need to solve equations of the form

$$f(x) = 0.^*$$



This means finding the values of x that satisfy the equation. Each such value of x is called a *root* of the equation $f(x) = 0$ and a *zero* of the function $f(x)$. These roots (or zeros) that are real numbers are called *real roots* (or real zeros). For many problems the roots of an equation can be determined analytically through algebraic manipulation; in many other instances, this is not possible. Numerical techniques can be used to estimate the roots when analytical methods are not suitable. When you use the [SOLVE] key on your HP-15C, you utilize an advanced numerical technique that lets you effectively and conveniently find *real roots* for a wide range of equations.†

Using [SOLVE]

In calculating roots, the [SOLVE] operation repeatedly calls up and executes a subroutine *that you write* for evaluating $f(x)$.

---

* Actually, *any* equation with one variable can be expressed in this form. For example, $f(x) = a$ is equivalent to $f(x) - a = 0$, and $f(x) = g(x)$ is equivalent to $f(x) - g(x) = 0$.

† The [SOLVE] function does not use the imaginary stack. Refer to the *HP-15C Advanced Functions Handbook* for information about complex roots.

The basic rules for using [SOLVE] are:

1. In Program mode, key in a subroutine that evaluates the function $f(x)$ that is to be equated to zero. This subroutine must begin with a label instruction ([f][LBL] *label*) and end up with a result for $f(x)$ in the X-register.

   In Run mode:

2. Key two initial estimates of the desired root, separated by [ENTER], into the X- and Y-registers. These estimates merely indicate to the calculator the approximate range of x in which it should initially seek a root of $f(x) = 0$.

3. Press [f][SOLVE] followed by the label of your subroutine. The calculator then searches for the desired zero of your function and displays the result. If the function that you are analyzing equals zero at more than one value of x, the routine will stop when it finds any one of those values. To find additional values, you can key in different initial estimates and use [SOLVE] again.

Immediately before [SOLVE] addresses your subroutine it places a value of x in the X-, Y-, Z-, and T-registers. This value is then used by your subroutine to calculate $f(x)$. Because the entire stack is filled with the x-value, this number is continually available to your subroutine. (The use of this technique is described on page 41).

Example: Use [SOLVE] to find the values of x for which

$$f(x) = x^2 - 3x - 10 = 0.$$

Using Horner's method (refer to page 79), you can rewrite $f(x)$ so that it is programmed more efficiently:

$$f(x) = (x - 3)x - 10.$$

In Program mode, key in the following subroutine to evaluate $f(x)$.

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

Keystrokes	Display		
f LBL 0	001-	42,21, 0	Begin with LBL instruction. Subroutine assumes stack loaded with x.
3	002-	3	
−	003-	30	Calculate $x - 3$.
×	004-	20	Calculate $(x - 3)x$.
1	005-	1	
0	006-	0	
−	007-	30	Calculate $(x - 3)x - 10$.
g RTN	008-	43 32	

In Run mode, key two initial estimates into the X- and Y-registers. Try estimates of 0 and 10 to look for a positive root.

Keystrokes	Display*		
g P/R			Run mode.
0 ENTER	0.0000	}	Initial estimates.
10	10		

You can now find the desired root by pressing f SOLVE 0. When you do this, the calculator will not display the answer right away. The HP-15C uses an iterative algorithm† to estimate the root. The algorithm analyzes your function by sampling it many times, perhaps a dozen times or more. It does this by repeatedly executing your subroutine. Finding a root will usually require about 30 seconds to 2 minutes; but sometimes the process will require even more time.

Press f SOLVE 0 and sit back while your HP-15C exhibits one of its powerful capabilities. The display flashes **running** while SOLVE is operating.

---

* Press f FIX 4 to obtain the displays shown here. The display setting does not influence the operation of SOLVE.

† An *algorithm* is a step-by-step procedure for solving a mathematical problem. An *iterative* algorithm is one containing a portion that is executed a number of times in the process of solving the problem.

Keystrokes	Display	
f SOLVE 0	5.0000	The desired root.

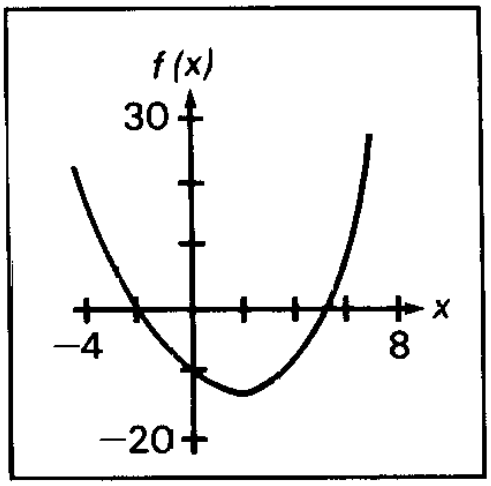
After the routine finds and displays the root, you can ensure that the displayed number is indeed a root of $f(x) = 0$ by checking the stack. You have seen that the display (X-register) contains the desired root. The Y-register contains a previous estimate of the root, which should be very close to the displayed root. The Z-register contains the value of your function evaluated at the displayed root.

Keystrokes	Display	
R↓	5.0000	A previous estimate of the root.
R↓	0.0000	Value of the function at the root showing that $f(x) = 0$.

Quadratic equations, such as the one you are solving, can have two roots. If you specify two new initial estimates, you can check for a second root. Try estimates of 0 and −10 to look for a negative root.

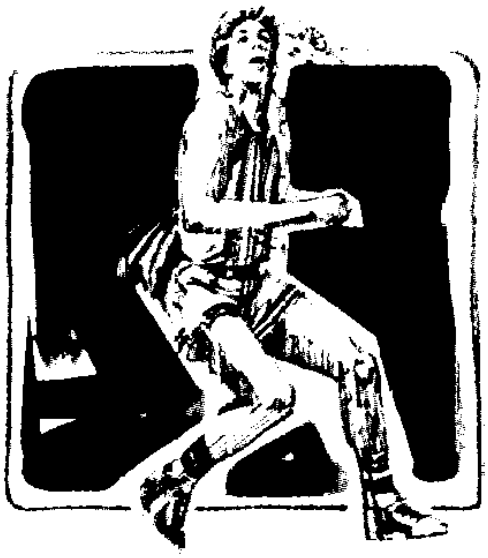
Keystrokes	Display	
0 ENTER	0.0000	Initial estimates.
10 CHS	−10	
f SOLVE 0	−2.0000	The second root.
R↓	−2.0000	A previous estimate of the root.
R↓	0.0000	Value of $f(x)$ at second root.

You have now found the two roots of $f(x) = 0$. Note that this quadratic equation *could* have been solved algebraically—and you would have obtained the same roots that you found using [SOLVE].



Graph of $f(x)$

The convenience and power of the [SOLVE] key become more apparent when you solve an equation for a root that cannot be determined algebraically.



Example: Champion ridget hurler Chuck Fahr throws a ridget with an upward velocity of 50 meters/second. If the height of the ridget is expressed as

$$h = 5000(1 - e^{-t/20}) - 200t,$$

how long does it take for it to reach the ground again? In this equation, h is the height in meters and t is the time in seconds.

Solution: The desired solution is the positive value of t at which $h = 0$.

Use the following subroutine to calculate the height.

Keystrokes	Display		
[g] [P/R]	000-		
[f] [LBL] [A]	001-	42,21,11	Begin with label.
2	002-	2	Subroutine assumes t is loaded in X- and Y-registers.
0	003-	0	
[÷]	004-	10	

Keystrokes	Display		
CHS	005-	16	$-t/20.$
e^x	006-	12	
CHS	007-	16	$-e^{-t/20}.$
1	008-	1	
+	009-	40	$1-e^{-t/20}.$
5	010-	5	
0	011-	0	
0	012-	0	
0	013-	0	
×	014-	20	$5000\,(1-e^{-t/20}).$
$x \gtrless y$	015-	34	Brings another t-value into X-register.
2	016-	2	
0	017-	0	
0	018-	0	
×	019-	20	$200t.$
−	020-	30	$5000\,(1-e^{-t/20})-200t.$
g RTN	021-	43 32	

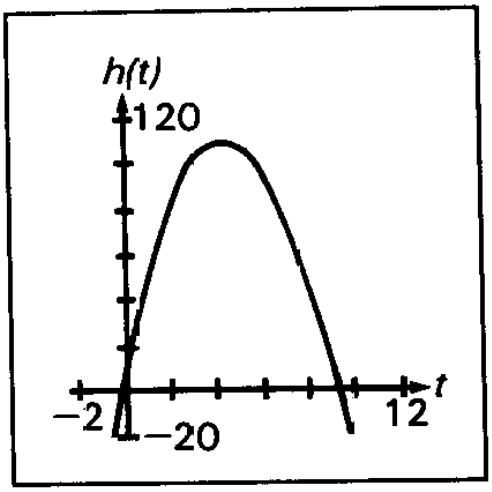
Switch to Run mode, key in two initial estimates of the time (for example, 5 and 6 seconds) and execute SOLVE.

Keystrokes	Display		
g P/R			Run mode.
5 ENTER	5.0000	}	Initial estimates.
6	6	}	
f SOLVE A	9.2843		The desired root.

Verify the root by reviewing the Y- and Z-registers.

Keystrokes	Display	
R↓	9.2843	A previous estimate of the root.
R↓	0.0000	Value of the function at the root showing that $h=0.$

Fahr's ridget falls to the ground 9.2843 seconds after he hurls it—a remarkable toss.



Graph of *h* versus *t*

When No Root Is Found

You have seen how the [SOLVE] key estimates and displays a root of an equation of the form $f(x) = 0$. However, it *is* possible that an equation has no real roots (that is, there is no real value of x for which the equality is true). Of course, you would not expect the calculator to find a root in this case. Instead, it displays **Error 8**.

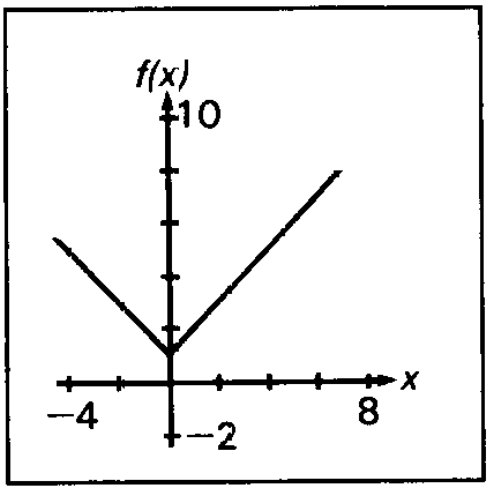
Example: Consider the equation

$$|x| = -1$$

which has no solution since the absolute value function is never negative. Express this equation in the required form

$$|x| + 1 = 0$$

and attempt to use [SOLVE] to find a solution.



Graph of $f(x) = |x| + 1$

Keystrokes	Display	
[g] [P/R]	000-	Program mode.
[f] [LBL] 1	001-42,21, 1	
[g] [ABS]	002- 43 16	
1	003- 1	
[+]	004- 40	
[g] [RTN]	005- 43 32	

Because the absolute-value function is minimum near an argument of zero, specify the initial estimates in that region, for instance 1 and −1. Then attempt to find a root.

Keystrokes	Display	
g P/R		Run mode.
1 ENTER	1.0000	Initial estimates.
1 CHS	−1	
f SOLVE 1	Error 8	This display indicates that no root was found.
←	0.0000	Clear error display.

As you can see, the HP-15C stopped seeking a root of $f(x) = 0$ when it decided that none existed—at least not in the general range of x to which it was initially directed. The **Error 8** display does not indicate that an "illegal" operation has been attempted; it merely states that no root was found where SOLVE presumed one might exist (based on your initial estimates).

If the HP-15C stops seeking a root and displays an error message, one of these three types of conditions has occurred:

- If repeated iterations all produce a constant nonzero value for the specified function, execution stops with the display **Error 8**.
- If numerous samples indicate that the *magnitude* of the function appears to have a nonzero minimum value in the area being searched, execution stops with the display **Error 8**.
- If an improper argument is used in a mathematical operation as part of your subroutine, execution stops with the display **Error 0**.

In the case of a constant function value, the routine can see no indication of a tendency for the value to move toward zero. This can occur for a function whose first 10 significant digits are constant (such as when its graph levels off at a nonzero horizontal asymptote) or for a function with a relatively broad, local "flat" region in comparison to the range of x-values being tried.

In the case where the function's magnitude reaches a nonzero minimum, the routine has logically pursued a sequence of samples for which the magnitude has been getting smaller. However, it has

not found a value of x at which the function's graph touches or crosses the x-axis.

The final case points out a potential deficiency in the subroutine rather than a limitation of the root-finding routine. Improper operations may sometimes be avoided by specifying initial estimates that focus the search in a region where such an outcome will not occur. However, the [SOLVE] routine is very aggressive and may sample the function over a wide range. It is a good practice to have your subroutine test or adjust potentially improper arguments prior to performing an operation (for instance, use [ABS] prior to [√x]). Rescaling variables to avoid large numbers can also be helpful.

The success of the [SOLVE] routine in locating a root depends primarily upon the nature of the function it is analyzing and the initial estimates at which it begins searching. The mere existence of a root does not ensure that the casual use of the [SOLVE] key will find it. If the function $f(x)$ has a nonzero horizontal asymptote or a local minimum of its magnitude, the routine can be expected to find a root of $f(x) = 0$ only if the initial estimates do not concentrate the search in one of these unproductive regions—and, of course, if a root actually exists.

Choosing Initial Estimates

When you use [SOLVE] to find the root of an equation, the two initial estimates that you provide determine the values of the variable x at which the routine begins its search. In general, the likelihood that you will find the particular root you are seeking increases with the level of understanding that you have about the function you are analyzing. Realistic, intelligent estimates greatly facilitate the determination of a root.

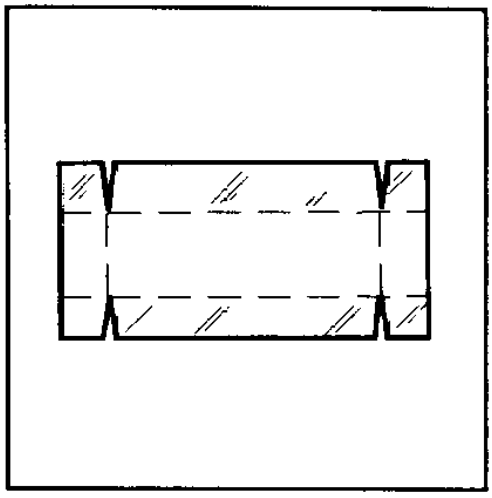
The initial estimates that you use may be chosen in a number of ways:

If the variable x has a limited range in which it is conceptually meaningful as a solution, it is reasonable to choose initial estimates within this range. Frequently an equation that is applicable to a real problem has, in addition to the desired solution, other roots that are physically meaningless. These usually occur because the equation being analyzed is appropriate only between

certain limits of the variable. You should recognize this restriction and interpret the results accordingly.

If you have some knowledge of the behavior of the function $f(x)$ as it varies with different values of x, you are in a position to specify initial estimates in the general vicinity of a zero of the function. You can also avoid the more troublesome ranges of x such as those producing a relatively constant function value or a minimum of the function's magnitude.

Example: Using a rectangular piece of sheet metal 4 decimeters by 8 decimeters, an open-top box having a volume of 7.5 cubic decimeters is to be formed. How should the metal be folded? (A taller box is preferred to a shorter one.)



Solution: You need to find the height of the box (that is, the amount to be folded up along each of the four sides) that gives the specified volume. If x is the height (or amount folded up), the length of the box is $(8 - 2x)$ and the width is $(4 - 2x)$. The volume V is given by

$$V = (8 - 2x)(4 - 2x)x.$$

By expanding the expression and then using Horner's method (page 79), this equation can be rewritten as

$$V = 4((x - 6)x + 8)x.$$

To get $V = 7.5$, find the values of x for which

$$f(x) = 4((x - 6)x + 8)x - 7.5 = 0.$$

The following subroutine calculates $f(x)$:

Keystrokes	Display		
g P/R	000-		Program mode.
f LBL 3	001-42,21,	3	Label.
6	002-	6	Assumes stack loaded with x.

Keystrokes	Display		
[−]	003–	30	
[×]	004–	20	$(x-6)x$.
8	005–	8	
[+]	006–	40	
[×]	007–	20	$((x-6)x+8)x$.
4	008–	4	
[×]	009–	20	$4((x-6)x+8)x$.
7	010–	7	
[·]	011–	48	
5	012–	5	
[−]	013–	30	
[g] [RTN]	014–	43 32	

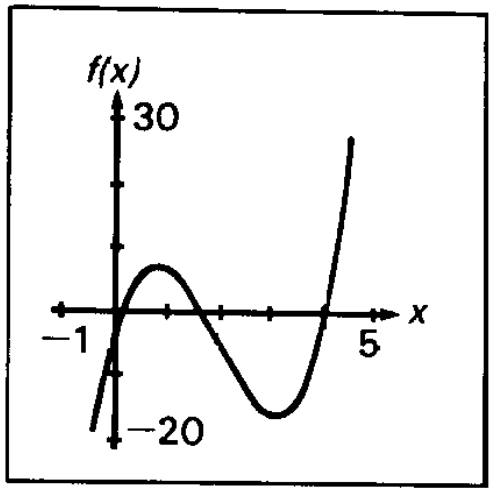
It seems reasonable that either a tall, narrow box or a short, flat box could be formed having the desired volume. Because the taller box is preferred, larger initial estimates of the height are reasonable. However, heights greater than 2 decimeters are not physically possible (because the metal is only 4 decimeters wide). Initial estimates of 1 and 2 decimeters are therefore appropriate.

Find the desired height:

Keystrokes	Display	
[g] [P/R]		Run mode.
1 [ENTER]	1.0000	Initial estimates.
2	2	
[f] [SOLVE] 3	1.5000	The desired height.
[R↓]	1.5000	Previous estimate.
[R↓]	0.0000	$f(x)$ at root.

By making the height 1.5 decimeters, a 5.0 × 1.0 × 1.5-decimeter box is specified.

If you ignore the upper limit on the height and use initial estimates of 3 and 4 decimeters (still less than the width), you will obtain a height of 4.2026 decimeters—a root that is physically meaningless. If you use small initial estimates such as 0 and 1 decimeter, you will obtain a height of 0.2974 decimeter—producing an undesirably short, flat box.



Graph of *f*(*x*)

As an aid for examining the behavior of a function, you can easily evaluate the function at one or more values of x using your subroutine in program memory. To do this, fill the stack with x. Execute the subroutine to calculate the value of the function (press f *letter label* or GSB *label*.

The values you calculate can be plotted to give you a graph of the function. This procedure is particularly useful for a function whose behavior you do not know. A simple-looking function may have a graph with relatively extreme variations that you might not anticipate. A root that occurs near a localized variation may be hard to find unless you specify initial estimates that are close to the root.

If you have no informed or intuitive concept of the nature of the function or the location of the zero you are seeking, you can search for a solution using trial-and-error. The success of finding a solution depends partially upon the function itself. Trial-and-error is often—but not always—successful.

- If you specify two moderately large positive or negative estimates and the function's graph does not have a horizontal asymptote, the routine will seek a zero which might be the most positive or negative (unless the function oscillates many times, as the trigonometric functions do).

- If you have already found a zero of the function, you can check for another solution by specifying estimates that are relatively distant from any known zeros.

- Many functions exhibit special behavior when their arguments approach zero. You can check your function to determine values of x for which any argument within your function becomes zero, and then specify estimates at or near those values.

Although two different initial estimates are usually supplied when using [SOLVE], you can also use [SOLVE] with the same estimate in both the X- and Y-registers. If the two estimates are identical, a second estimate is generated internally. If your single estimate is nonzero, the second estimate differs from your estimate by one count in the seventh significant digit. If your estimate is zero, 1×10^{-7} is used as the second estimate. Then the root-finding procedure continues as it normally would with two estimates.

Using [SOLVE] in a Program

You can use the [SOLVE] operation as part of a program. Be sure that the program provides initial estimates in the X- and Y-registers just prior to the [SOLVE] operation. The [SOLVE] routine stops with a value of x in the X-register and the corresponding function value in the Z-register. If the x-value is a root, the program proceeds to the next line. If the x-value is not a root, the next line is skipped. (Refer also to Interpreting Results on page 226 for a further explanation of roots.) Essentially, the [SOLVE] instruction tests whether the x-value is a root and then proceeds according to the "Do if True" rule. The program can then handle the case of not finding a root, such as by choosing new initial estimates or changing a function parameter.

The use of [SOLVE] as an instruction in a program utilizes one of the seven pending returns in the calculator. Since the subroutine called by [SOLVE] utilizes another return, there can be only five other pending returns. Executed from the keyboard, on the other hand, [SOLVE] itself does not utilize one of the pending returns, so that six pending returns are available for subroutines within the subroutine called by [SOLVE]. Remember that if all seven pending returns have been utilized, a call to another subroutine will result in a display of **Error 5**. (Refer to page 105.)

Restriction on the Use of [SOLVE]

The one restriction regarding the use of [SOLVE] is that [SOLVE] cannot be used recursively. That is, you cannot use [SOLVE] in a subroutine that is called during the execution of [SOLVE]. If this situation occurs, execution stops and **Error 7** is displayed. It *is* possible, however, to use [SOLVE] with [f], thereby using the advanced capabilities of both of these keys.

Memory Requirements

[SOLVE] requires five registers to operate. (Appendix C explains how they are automatically allocated from memory.) If five unoccupied registers are not available, [SOLVE] will not run and **Error 10** will be displayed.

A routine that combines [SOLVE] and [f] requires 23 registers of space.

For Further Information

In appendix D, Advanced Use of [SOLVE], additional techniques and explanations for using [SOLVE] are presented. These include:

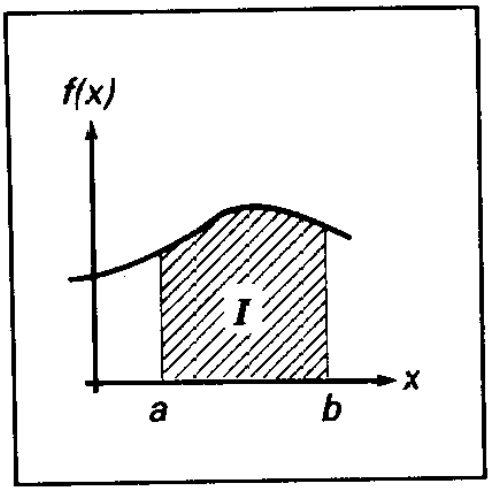
- How [SOLVE] works.
- Accuracy of the root.
- Interpreting results.
- Finding several roots.
- Limiting estimation time.

Section 14

Numerical Integration

Many problems in mathematics, science, and engineering require calculating the definite integral of a function. If the function is denoted by $f(x)$ and the interval of integration is a to b, the integral can be expressed mathematically as

$$I=\int_a^b f(x)\,dx.$$



The quantity I can be interpreted geometrically as the area of a region bounded by the graph of $f(x)$, the x-axis, and the limits $x=a$ and $x=b$.*

When an integral is difficult or impossible to evaluate by analytical methods, it can be calculated using numerical techniques. Usually, this can be done only with a fairly complicated computer program. With your HP-15C, however, you can easily do numerical integration using the $\boxed{\int_y^x}$ (*integrate*) key.†

Using $\boxed{\int_y^x}$

The basic rules for using $\boxed{\int_y^x}$ are:

1. In Program mode, key in a subroutine that evaluates the function $f(x)$ that you want to integrate. This subroutine must begin with a label instruction ($\boxed{f}\boxed{LBL}$ *label*) and end up with a value for $f(x)$ in the X-register.

* Provided that $f(x)$ is nonnegative throughout the interval of integration.

† The $\boxed{\int_y^x}$ function does not use the imaginary stack. Refer to the *HP-15C Advanced Functions Handbook* for information about using $\boxed{\int_y^x}$ in Complex mode.

In Run mode:

2. Key the lower limit of integration (a) into the X-register, then press [ENTER] to lift it into the Y-register.
3. Key the upper limit of integration (b) into the X-register.
4. Press [f] [∫] followed by the label of your subroutine.

Example: Certain problems in physics and engineering require calculating *Bessel functions*. The Bessel function of the first kind of order 0 can be expressed as

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x \sin\theta)\, d\theta.$$

Find

$$J_0(1) = \frac{1}{\pi}\int_0^{\pi} \cos(\sin\theta)\, d\theta.$$

In Program mode, key in the following subroutine to evaluate the function $f(\theta) = \cos(\sin\theta)$.

Keystrokes	Display	
[g] [P/R]	000-	Program mode.
[f] CLEAR [PRGM]	000-	Clear program memory.
[f] [LBL] 0	001-42,21, 0	Begin subroutine with a [LBL] instruction. Subroutine assumes a value of θ is in X-register.
[SIN]	002- 23	Calculate sin θ.
[COS]	003- 24	Calculate cos (sin θ).
[g] [RTN]	004- 43 32	

Now, in Run mode key the lower limit of integration into the Y-register and the upper limit into the X-register. For this particular problem, you also need to specify Radians mode for the trigonometric functions.

Keystrokes	Display	
g P/R		Run mode.
0 ENTER	0.0000	Key lower limit, 0, into Y-register.
g π	3.1416	Key upper limit, π, into X-register.
g RAD	3.1416	Specify Radians mode for trigonometric functions.

Now you are ready to press f ∫ 0 to calculate the integral. When you do so, you'll find that—just as with SOLVE—the calculator will not display the result right away, as it does with other operations. The HP-15C calculates integrals using a sophisticated iterative algorithm. Briefly, this algorithm evaluates $f(x)$, the function to be integrated, at many values of x between the limits of integration. At each of these values, the calculator evaluates the function by executing the subroutine you write for that purpose. When the calculator must execute the subroutine many times—as it does when you press ∫—you can't expect any answer right away. Most integrals will require on the order of 30 seconds to 2 minutes; but some integrals will require even more. Later on we'll discuss how you can decrease the time somewhat; but for now, press f ∫ 0 and take a break (or read ahead) while the HP-15C takes care of the drudgery for you.

Keystrokes	Display	
f ∫ 0	2.4040	$= \int_0^{\pi} \cos(\sin\theta)\, d\theta.$

In general, don't forget to multiply the value of the integral by whatever constants, if any, are outside the integral. In this particular problem, we need to multiply the integral by $1/\pi$ to get $J_0(1)$:

Keystrokes	Display	
g π	3.1416	
÷	0.7652	$J_0(1)$.

Before calling the subroutine you provide to evaluate $f(x)$, the [∫f] algorithm—just like the [SOLVE] algorithm—places the value of x in the X-, Y-, Z-, and T-registers. Because every stack register contains the x-value, your subroutine can calculate with this number without having to recall it from a storage register. The subroutines in the next two examples take advantage of this feature. (A polynomial evaluation technique that assumes the stack is filled with the value of x is discussed on page 79.)

Note: Since the calculator puts the value of x into all stack registers, any numbers previously there will be replaced by x. Therefore, if the stack contains intermediate results that you'll need after you calculate an integral, store those numbers in storage registers and recall them later.

Occasionally you may want to use the subroutine that you wrote for the [∫f] operation to merely evaluate the function at some value of x. If you do so with a function that gets x from the stack more than once, be sure to fill the stack manually with the value of x, by pressing [ENTER] [ENTER] [ENTER], before you execute the subroutine.

Example: The Bessel function of the first kind of order 1 can be expressed as

$$J_1(x) = \frac{1}{\pi}\int_0^{\pi} \cos(\theta - x\sin\theta)\, d\theta.$$

Find

$$J_1(1) = \frac{1}{\pi}\int_0^{\pi} \cos(\theta - \sin\theta)\, d\theta.$$

Key in the following subroutine that evaluates the function $f(\theta) = \cos(\theta - \sin\theta)$.

Keystrokes	Display	
[g] [P/R]	000–	Program mode.
[f] [LBL] 1	001–42,21, 1	Begin subroutine with a label.

Keystrokes	Display		
SIN	002–	23	Calculate sin θ.
−	003–	30	Since a value of θ will be placed into the Y-register by the $\int$ algorithm before it executes this subroutine, the − operation at this point will calculate $(\theta - \sin\theta)$.
COS	004–	24	Calculate cos $(\theta - \sin\theta)$.
g RTN	005–	43 32	

In Run mode, key the limits of integration into the X- and Y-registers. Be sure that the trigonometric mode is set to Radians, then press f $\int$ 1 to calculate the integral. Finally, multiply the integral by $1/\pi$ to calculate $J_1(1)$.

Keystrokes	Display	
g P/R		Run mode.
0 ENTER	0.0000	Key lower limit into Y-register.
g π	3.1416	Key upper limit into X-register.
g RAD	3.1416	(If not already in Radians mode.)
f $\int$ 1	1.3825	$= \int_0^{\pi} \cos(\theta - \sin\theta)\, d\theta$.
g π ÷	0.4401	$J_1(1)$.

Example: Certain problems in communications theory (for example, pulse transmission through idealized networks) require calculating an integral (sometimes called the *sine integral*) of the form

$$Si(t) = \int_0^t \frac{\sin x}{x}\, dx.$$

Find $Si(2)$.

Key in the following subroutine to evaluate the function $f(x) = (\sin x)/x$.*

Keystrokes	Display	
[g] [P/R]	000-	Program mode.
[f] [LBL] .2	001-42,21,.2	Begin subroutine with a [LBL] instruction.
[SIN]	002- 23	Calculate sin x.
[$x \gtrless y$]	003- 34	Since a value of x will be placed in the Y-register by the [f] algorithm before it executes this subroutine, the [$x \gtrless y$] operation at this point will return x to the X-register and move sin x to the Y-register.
[÷]	004- 10	Divide sin x by x.
[g] [RTN]	005- 43 32	

Now key the limits of integration into the X- and Y-registers. In Radians mode, press [f] [f] .2 to calculate the integral.

Keystrokes	Display	
[g] [P/R]	0.4401	Run mode
0 [ENTER]	0.0000	Key lower limit into Y-register.
2	2	Key upper limit into X-register.
[g] [RAD]	2.0000	(If not already in Radians mode.)
[f] [f] .2	1.6054	$Si(2)$.

* If the calculator attempted to evaluate $f(x) = (\sin x)/x$ at $x = 0$, the lower limit of integration, it would terminate with **Error 0** in the display (signifying an attempt to divide by zero), and the integral could not be calculated. However, the [f] algorithm normally does *not* evaluate functions at either limit of integration, so the calculator *can* calculate the integral of a function that is undefined there. Only when the endpoints of the interval of integration are extremely close together, or the number of sample points is extremely large, does the algorithm evaluate the function at the limits of integration.

Accuracy of [∫]

The accuracy of the integral of any function depends on the accuracy of the function itself. Therefore, the accuracy of an integral calculated using [∫] is limited by the accuracy of the function calculated by your subroutine.* To specify the accuracy of the function, set the display format so that the display shows *no more* than the number of digits that you consider accurate in the function's values.† If you specify fewer digits, the calculator will compute the integral more quickly;‡ but it will presume that the function is accurate to only the number of digits specified in the display format. We'll show you how you can determine the accuracy of the calculated integral after we say another word about the display format.

You'll recall that the HP-15C provides three types of display formating: [FIX], [SCI], and [ENG]. Which display format should be used is largely a matter of convenience, since for many integrals you'll get about the same results using any of them (provided that the number of digits is specified correctly, considering the magnitude of the function). Because it's more convenient to use [SCI] display format when calculating most integrals, we'll use [SCI] when calculating integrals in subsequent examples.

Note: Remember that once you have set the display format, you can change the number of digits appearing in the display by storing a number in the Index register and then pressing [f] [FIX] [I], [f] [SCI] [I], or [f] [ENG] [I], as described in section 10. This capability is especially useful when [∫] is executed as part of a program.

---

* It is possible that integrals of functions with certain characteristics (such as spikes or very rapid oscillations) *might* be calculated inaccurately. However, *this possibility is very small*. The general characteristics of functions that could cause problems, as well as techniques for dealing with them, are discussed in appendix E.

† The accuracy of a calculated function depends on such considerations as the accuracy of empirical constants in the function as well as round-off error in the calculations. These considerations are discussed in more detail in the *HP-15C Advanced Functions Handbook*.

‡ The reason for this is discussed in appendix E.

Because the accuracy of any integral is limited by the accuracy of the function (as indicated in the display format), the calculator cannot compute the value of an integral exactly, but rather only *approximates* it. The HP-15C places the uncertainty* of an integral's approximation in the Y-register at the same time it places the approximation in the X-register. To determine the accuracy of an approximation, check its uncertainty by pressing [x≷y].

Example: With the display format set to [SCI] 2, calculate the integral in the expression for $J_1(1)$ (from the example on page 197).

Keystrokes	Display		
0 [ENTER]	**0.0000**		Key lower limit into Y-register.
[g] [π]	**3.1416**		Key upper limit into X-register.
[g] [RAD]	**3.1416**		(If not already in Radians mode.)
[f] [SCI] 2	**3.14**	**00**	Set display format to [SCI] 2.
[f] [∫y] 1	**1.38**	**00**	Integral approximated in [SCI] 2.
[x≷y]	**1.88**	**-03**	Uncertainty of [SCI] 2 approximation.

The integral is 1.38 ± 0.00188. Since the uncertainty would not affect the approximation until its third decimal place, you can consider all the displayed digits in this approximation to be accurate. In general, though, it is difficult to anticipate how many

* No algorithm for numerical integration can compute the exact difference between its approximation and the actual integral. But the algorithm in the HP-15C estimates an "upper bound" on this difference, which is the *uncertainty* of the approximation. For example, if the integral $Si(2)$ is 1.6054 ± 0.0001, the approximation to the integral is 1.6054 and its uncertainty is 0.0001. This means that while we don't know the exact difference between the actual integral and its approximation, we *do* know that it is highly unlikely that the difference is bigger than 0.0001. (Note the first footnote on page 200.)

digits in an approximation will be unaffected by its uncertainty. This depends on the particular function being integrated, the limits of integration, and the display format.

If the uncertainty of an approximation is larger than what you choose to tolerate, you can decrease it by specifying a greater number of digits in the display format and repeating the approximation.*

Whenever you want to repeat an approximation, you don't need to key the limits of integration back into the X- and Y-registers. After an integral is calculated, not only are the approximation and its uncertainty placed in the X- and Y-registers, but in addition the upper limit of itegration is placed in the Z-register, and the lower limit is placed in the T-register. To return the limits to the X- and Y-registers for calculating an integral again, simply press [R↓] [R↓].

Example: For the integral in the expression for $J_1(1)$, you want an answer accurate to four decimal places instead of only two.

Keystrokes	Display		
[f] [SCI] 4	1.8826	-03	Set display format to [SCI] 4.
[R↓] [R↓]	3.1416	00	Roll down stack until upper limit appears in X-register.
[f] [∫] 1	1.3825	00	Integral approximated in [SCI] 4.
[x≷y]	1.7091	-05	Uncertainty of [SCI] 4 approximation.

The uncertainty indicates that this approximation is accurate to at least four decimal places. Note that the uncertainty of the [SCI] 4 approximation is about one-hundredth as large as the uncertainty of the [SCI] 2 approximation. In general, the uncertainty of any [∫] approximation decreases by about a factor of 10 for each additional digit specified in the display format.

---

* Provided that $f(x)$ is still calculated accurately to the number of digits shown in the display.

In the preceding example, the uncertainty indicated that the approximation *might* be correct to only four decimal places. If we temporarily display all 10 digits of the approximation, however, and compare it to the actual value of the integral (actually, an approximation known to be accurate to a sufficient number of decimal places), we find that the approximation is actually more accurate than its uncertainty indicates.

Keystrokes	Display	
[x≷y]	1.3825 00	Return approximation to display.
[f] CLEAR [PREFIX]	1382459676	All 10 digits of approximation.

The value of this integral, correct to eight decimal places, is 1.38245969. The calculator's approximation is accurate to *seven* decimal places rather than only four. In fact, since the uncertainty of an approximation is calculated very conservatively, *the calculator's approximation in most cases will be more accurate than its uncertainty indicates*. However, normally there is no way to determine *just how accurate* an approximation is.

For a more detailed look at the accuracy and uncertainty of [∫] approximations, refer to appendix E.

Using [∫] in a Program

[∫] can appear as an instruction in a program provided that the program is not called (as a subroutine) by [∫] itself. In other words, [∫] cannot be used recursively. Consequently, you cannot use [∫] to calculate multiple integrals; if you attempt to do so, the calculator will halt with **Error 7** in the display. However, [∫] *can* appear as an instruction in a subroutine called by [SOLVE].

The use of [∫] as an instruction in a program utilizes one of the seven pending returns in the calculator. Since the subroutine called by [∫] utilizes another return, there can be only five other pending returns. Executed from the keyboard, on the other hand, [∫] itself does not utilize one of the pending returns, so that six pending returns are available for subroutines within the subroutine called

by [∫f]. Remember that if all seven pending returns have been utilized, a call to another subroutine will result in a display of **Error 5**. (Refer to page 105.)

Memory Requirements

[∫f] requires 23 registers to operate. (Appendix C explains how they are automatically allocated from memory.) If 23 unoccupied registers are not available, [∫f] will not run and **Error 10** will be displayed.

A routine that combines [∫f] and [SOLVE] also requires 23 registers of space.

For Further Information

This section has given you the information you need to use [∫f] with confidence over a wide range of applications. In appendix E, more esoteric aspects of [∫f] are discussed. These include:

- How [∫f] works.
- Accuracy, uncertainty, and calculation time.
- Uncertainty and the display format.
- Conditions that could cause incorrect results.
- Conditions that prolong calculation time.
- Obtaining the current approximation to an integral.