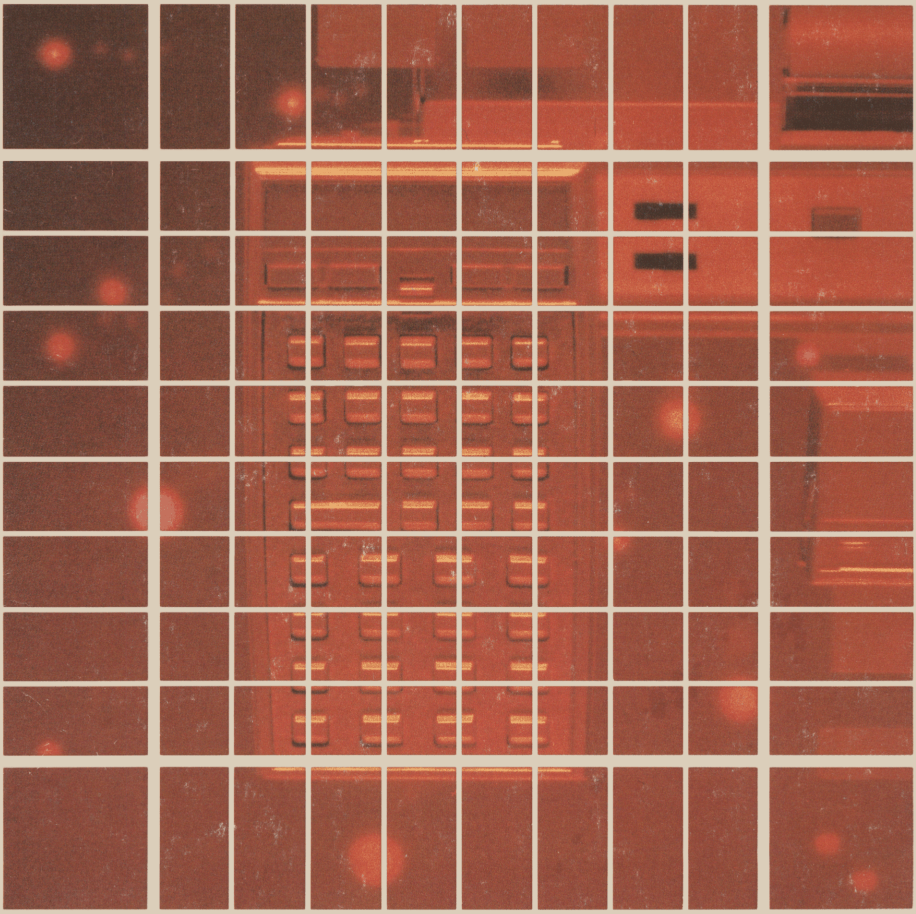
HEWLETT-PACKARD

HP-41 ADVANTAGE

ADVANCED SOLUTIONS PAC



KEYSTROKE NOTATION USED IN THIS MANUAL

Notation Example	Description
[Σ+]	A keyboard function. Press [Σ+].
[Σ−]	A shifted keyboard function. Press ■ [Σ+] (sequentially, not simultaneously).
[A] (**FX**)	A customized function for a particular program. Press [Σ+]. (Corresponds to key with blue letter "A".) **FX** is the display's menu label for [A] in this example.
[TVM]	A non-keyboard function. To execute it, press [XEQ] [ALPHA] TVM [ALPHA]. Alternatively, you can assign this function to a User key and then execute it as a single key.
[SIZE] 013	[XEQ] [ALPHA] SIZE [ALPHA] [0] [1] [3]
ABC	Alpha-keyboard characters mapped to the blue letters on the keys. Press [ALPHA] to start and finish.
123	Shifted Alpha-keyboard characters (mapped as shown on the back label of the calculator).

HEWLETT PACKARD

The HP-41 Advantage

Advanced Solutions Pac

July 1985

00041-90546

Printed in U.S.A.

ACKNOWLEDGMENTS

The matrix operations in this pac were based on the CCD ROM, written by W & W Software Products GmbH, 5060 Bergisch Gladbach 2, West Germany.

The root-finding and numerical-integration routines in this pac were adapted from those in the HP-15C by Firmware Specialists, Inc., 605 NW 5th Street #2A, Corvallis, Oregon 97330.

The original concept for the content and user interface of this pac was developed by Chris Bunsen, Corvallis, Oregon.

NOTICE

TABLE OF CONTENTS

INTRODUCTION

The HP–41 Advantage—the Advanced Solutions Pac—gives you a selection of programs and functions for solving advanced mathematical and engineering problems, curve-fitting statistical problems, and simple financial problems (the time value of money). It's a broad, powerful solution set for the technical student or professional.

Many of the routines used internally by this pac have been made accessible to you for use as subroutines in your own programs.

This manual provides a description of each program or function set with relevant equations, step-by-step instructions for operation, examples with the keystrokes needed for the solution, and descriptions of the user-accessible subroutines.

> **Note:** Before plugging in your HP-41 Advantage Pac, *turn the calculator off*, and be sure you understand the section "Inserting and Removing Application Modules".

INSERTING AND REMOVING APPLICATION MODULES

Before inserting an application module for the first time, familiarize yourself with the following information.

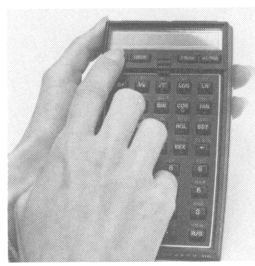
Up to four application modules can be plugged into the ports in the HP-41. The names of all programs contained in an inserted module are displayed in catalog 2 (CATALOG 2).

CAUTION

> Always turn the calculator off before inserting or removing any plug-in accessories. Otherwise, both the calculator and the accessory could be damaged.

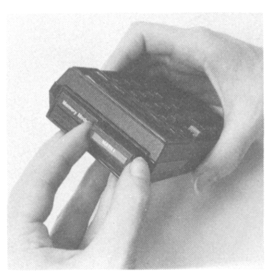
To insert a module:

Turn the calculator off!



Remove the port cover. Save it to cover the port when it is empty.

In an HP-41C, insert the application module into any port *after* the last memory module. (HP-41CV's and HP-41CX's don't use memory modules.) Insert the module with its label right-side up, as shown. For example, if you have a memory module in port 1, you can insert an application module into port 2, 3, or 4. (The port numbers are diagrammed on the upper back of the calculator.) *Never insert an application module into a port with a smaller number than a memory module's port.*



Plug in any additional application modules, also after the last memory module. Cover any unused ports.

The application module programs are now ready to use!

To remove a module:

Turn the calculator off! (Failure to do so could damage both the calculator and the module.)

Grasp the desired module handle and pull it out as shown.

Cover the empty port with a port cover.

Any other plug-in accessories (such as the HP 82104A Card Reader or the HP 82153A Wand) should be plugged in like application modules.

You can leave gaps in the port sequence. For example, you could plug a memory module into port 1 and an application module into port 4, leaving ports 2 and 3 empty.

HOW TO USE THIS MANUAL AND PAC

What Is in Each Chapter

Each chapter in this manual covers a different program or set of functions. With the exception of the two chapters on matrices, each chapter is independent of the others.

Starting each chapter is a **description** of the purpose of its program or functions. The **equations** on which the program is based are given and, if appropriate, **references** for further information are noted. Where appropriate, the **valid range for data values** is given. In some cases, the program will work outside its range of validity, but the result might not be accurate enough for you.

The Instruction Table

The heart of each chapter is its **instructions** and **instruction table**. This gives you general and step-by-step instructions for using the program or functions. It tells you what kinds of data values to key into the calculator, and which keys to press to compute results.

At its head, the instruction table specifies the minimum number of data-storage registers needed to run the program. (Refer to "Allocating Registers," below.)

		Size: 016
Instructions	**Key In:**	**Display**
⋮ 3. Input your data pairs: Repeat for each pair. ⋮	*y* [ENTER] *x* [A] (Σ+)	Σ+ **CLΣ FIT**

This column describes what you need to do, including what kind of input (data values) to key in.

This column tells you which key(s) to press to enter your input or compute a result.

This column shows you what you should see in the calculator's display after you follow the given instruction. The display most often shows a result, a prompt for information, or a menu. (The *menu interface* used by many programs in this pac is described in "Using the Menu Interface".)

Following the instruction table are **remarks** about the program—details of its operation and clarification of certain points.

Each chapter has **examples** for using its program or functions.

Lastly is **programming information** for calling the user-accessible subroutines within the given program for use in programs you might write.

Allocating Registers ([SIZE])

The instruction table tells you the minimum number of data-storage registers required to run a specific program. To allocate these *nnn* storage registers, use the [SIZE] function (press [XEQ] [ALPHA] SIZE [ALPHA] *nnn*). For more information on this function, refer to the owner's manual for the HP-41. If you try to run a program but get the message

SIZE>=*nnn*

you need to set the [SIZE] to (at least) *nnn*. Then press [R/S] to continue the program.

Notation for Calculator Keys

As explained in the owner's manual for the HP-41, the HP-41 has both *keyboard* and *nonkeyboard* functions. These two types of functions are invoked (*executed*) in two different ways. Keyboard functions have their own keys on the keyboard (such as [TAN] and [x^2]). Nonkeyboard functions—*including programs*—must have their names (also called *Alpha names*) typed into the display after pressing [XEQ].*

Notation Example	Keys to Press
[Σ+]	[Σ+] This is a keyboard function.
[Σ−]	■ [Σ+] (Press these sequentially, not simultaneously.) This is a *shifted* keyboard function.
[A] (**FX**)	[Σ+] (Σ+ is printed on the top surface; A is printed on the forward face.) This is a "customized" function for a particular program. **FX** is what would appear (for example) in the display above [A]. **FX** is the *menu label* for [A].
[XEQ] [TVM]	[XEQ] [ALPHA] TVM [ALPHA] (The [ALPHA] key toggles the Alpha keyboard on and off.) This is a *non*-keyboard function. It can also be executed as a User key. (Refer to the owner's manual for the HP-41.)
[XEQ] [SIZE] 013	[XEQ] [ALPHA] SIZE [ALPHA] [0] [1] [3]

This pac uses keys in the top two rows as special, redefined functions. They are represented then as [A] through [J], not as [Σ+], etc.

Using the Menu Interface

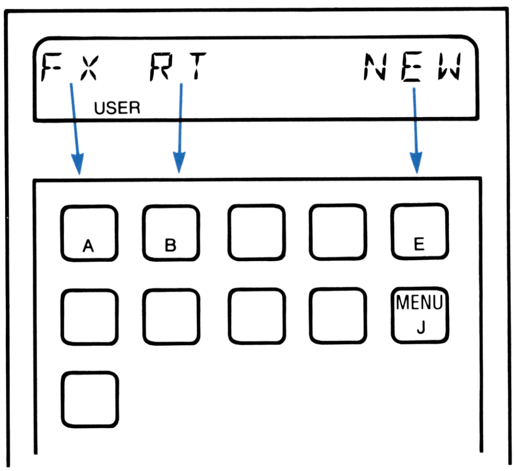
This pac supplies both new *functions* and *programs* for your use. Each chapter explains what is available. The individual functions operate like other HP-41 nonkeyboard functions. The programs are more sophisticated *and* easier to use: they combine several new functions plus a *user interface* with prompts and menus that lead you through data input and the resulting output.

---

* The other, faster alternative for executing a nonkeyboard function or program is to assign its name to a key on the User keyboard. Refer to the owner's manual for the HP-41.

Most important, each program redefines some keys in the top two rows of the calculator to perform (with a single keystroke) operations defined in the program. *For a program to work as given, you must clear any existing User-key assignments in the top two rows.* To use these redefined keys, the User keyboard must be active. All of the programs in this pac that provide this feature *automatically* activate the User keyboard when they are started. If you deactivate the User keyboard for any reason, you must reactivate it (press [USER]) to use the redefined keys. The menus have *labels* indicating the identities of those redefined keys. Here is an example:

The Menu for Polynomial Solutions



The **FX** menu label shows you that when you run PLY, the top left key on the calculator, [A], is redefined to evaluate the polynomial $f(x)$ at x. [B] (identified by the label **RT**) is redefined to compute the root, and [E] (identified by the label **NEW**) initializes the program to accept a new polynomial.

The [J] key has special significance. In all menus in this pac, pressing [J] has the effect of recalling the menu to the display.* You can do this as often and whenever you like: the menu is simply an aid in identifying keys.

Error Messages. Should you get an error message during a program, it is handy to press [J] (after remedying the error condition) to display the menu again. For a definition of error messages, refer to your HP-41 owner's manual. Error messages that can occur with matrix operations are described in the chapter "Matrix Functions."

If the Calculator Turns Off. If the calculator turns off while you are working with a program, you will find the display changed when you turn it back on. The display will show the X-register, without any prompts or menu that might have been in the display before the calculator went off. The program is still active, but it is best to re-start it by pressing [J] to recall the menu.

If you are running a program that does *not* have a menu, it might be necessary to set flag 21 to re-establish proper display of results.

The Program-Execution Indicator

If you are not already familiar with the program-execution indicator (⊁), you soon will be. It appears and moves across the display whenever a program is actively running. So if you perform an operation from a program, the moving indicator shows you that the calculation is in process.

Listing the Contents of the Module

Catalog 2 shows you the names of all programs and subroutines in this pac (and any other modules plugged in). Press [CATALOG] 2.

Using Programs as Subroutines

You can call the programs (and some subprograms) in this pac as subroutines for your own programs in the HP-41's memory. Refer to the section on "Programming Information" at the end of many chapters.

---

* The [J] key is the same as the [TAN] key. We use the letter designations so as not to confuse the "old" function (tangent) with the new one.

Using a Printer

If you have a printer plugged into the HP-41 as you use this pac, set it to MAN mode for the most readable automatic print-out of your inputs and results. (Some programs require NORMAL mode.) NORMAL mode lists all input values and keystrokes you use, as well.

Copying Programs from the Pac

Many of the programs in this pac are copiable using the [COPY] function. However, *it is not necessary to copy a program into main memory in order to use it.* Also, it is not necessary to copy a subroutine in order to gain access to it for a program of your own.

Using Labels

You should avoid using labels in your own programs that are identical to labels in this application pac. In case of a label conflict, the label within program memory has priority over the label within the application pac. All program labels used in this pac are listed in catalog 2.

Conflicts with Other Application Modules

> **Note:** Do not have both the HP-41 Advantage Advanced Solutions Pac and the HP-IL Development Module plugged into the HP-41 at the same time. These two modules share the same ROM identification numbers, and using them together will cause problems with the operation of the calculator.

Certain function names used by the HP-41 Advantage are also used by the HP-41 Math Pac and the HP-41 Real Estate Pac. When using these functions, you should remove the modules whose functions you do not want accessed.

Duplicate Functions

Math Pac	Real Estate Pac
All complex-number functions.	N, PV, PMT, FV, and *I
All functions in DIFEQ.	

Getting Help

If you have questions regarding the operation of the calculator, be sure to refer to the owner's manual for the HP-41 for information. If you have technical problems with this pac that the manual cannot resolve, you can call or write Hewlett-Packard for technical customer assistance. Refer to your HP-41 owner's manual for the address and telephone number.

THE MATRIX PROGRAM

The Advantage Pac provides extensive capabilities for creating, storing, and calculating with real or complex matrices. This functionality is available to you as either *individual functions* or as a *program* with menus and prompts. This is the case with many of the other subject areas in this pac. However, unlike the other topics, the topic of matrices is here divided into two separate chapters because of its size and complexity.

This chapter describes the matrix program, MATRX—the easy, "user-friendly" way to use the most common matrix operations on a newly created matrix. To use MATRX you do not need to know how the calculator stores and treats matrices in its memory. The next chapter, "The Matrix Functions", lists and defines every matrix function in the pac, including those called by MATRX. Using these functions on their own requires a more intimate knowledge of how and where the calculator stores matrices.

What This Program Can Do

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 vaues of x_1 and x_2. These equations can be expressed in matrix form as $\mathbf{AX} = \mathbf{B}$, where

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

A is the *coefficient matrix* for the system, **B** is the *column* or *constant matrix*, and **X** is the *solution* or *result matrix.*

For such a matrix system, the MATRX program creates (dimensions) a square real or complex matrix, **A**, and a column matrix, **B**. You can then:

- Enter, change ("edit"), or just view elements in **A** and **B**.
- Invert **A**.
- Transpose **A** if **A** is real.

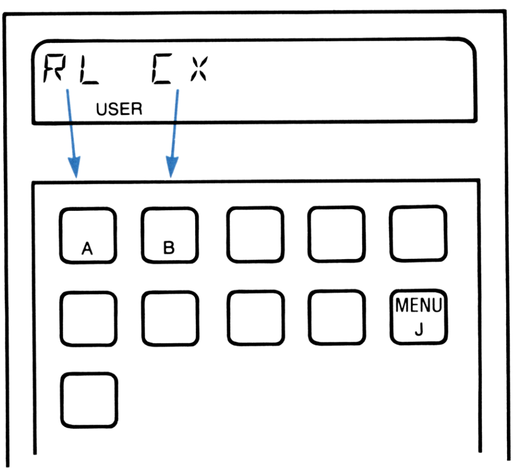
- Find the determinant of **A** if **A** is real.
- Solve the system of simultaneous equations by finding the solution to **AX** = **B**.

The size of your matrix is limited only by available memory. (Each real matrix requires one register plus one register for each element.) If you want to store more than one matrix, you will need to use the matrix function [MATDIM], described in the next chapter. The MATRX program does not store or recall matrices; it works with a single square matrix **A** and a single column matrix **B**. When you enter new elements into **A** you destroy its old elements.

Instructions

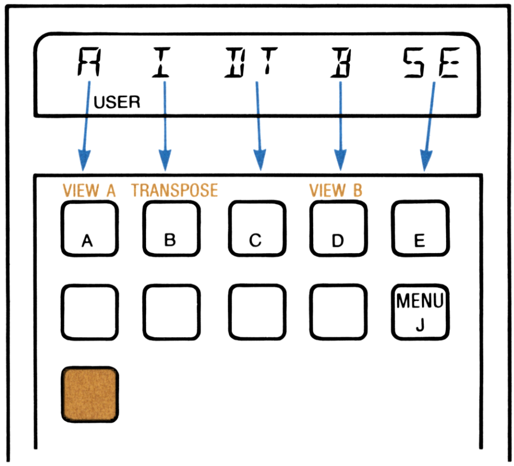
MATRX has two menus to show you which key corresponds to which function. The *initial menu* you see is to select a real or complex matrix:

Initial Menu



After you make this selection, input the order of the matrix, and press [R/S], you will see the *main menu*:

Main Menu



This menu shows you the choice of matrix operations you have in MATRX. Press [J] to recall this menu to the display at any time. This will not disturb the program in any way.

To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then recall the menu by pressing [J]. (However, you do not *need* to clear the program's display before performing calculations.)

- The program starts by asking you for a new matrix. It has you specify real vs. complex and the order (dimension) of a square matrix for **A**.
- The program does *not* clear previous matrix data, so previous data—possibly meaningless data—will fill your new matrices **A** and **B** until you enter new values for their elements.
- Each element of a complex matrix has two values (a real part and an imaginary part) and requires four times as much memory to store as an element in a real matrix. The prompts for real parts x_{11}, x_{12}, etc. are **1:1= ?**, **1:2= ?**, etc. The prompts for complex parts $x_{11} + iy_{11}$, $x_{12} + iy_{12}$, etc. are **RE.1:1= ?**, **IM.1:1= ?**, **RE.1:2= ?**, **IM.1:2= ?**, etc.

The next chapter ("Matrix Functions") includes a complete discussion under "How a Matrix Is Stored" of the specific requirements for matrix storage. You do not need to figure this out in order to use this program, however, because the program prompts you for the proper memory setting with the message **SIZE>=*nnn*** if your memory size is not large enough. (You would then execute [SIZE] *nnn* to size memory adequately.)

The following table shows the keystrokes to execute matrix operations in the MATRX program. All of these operations are also available as individual HP-41 functions, described in the next chapter.

Instruction Table for MATRX

Instructions	Key In:	Display Size: variable*
1. Start program MATRX.	[XEQ] [MATRX]†	**RL CX**
2. Select a new real (**RL**) or complex (**CX**) matrix.	[A] (**RL**) or [B] (**CX**)	**ORDER=?**
3. Enter dimension *n* of your square matrix, A.	*n* [R/S]	**A I DT B SE**
4. Enter the elements of your matrix A. The **?** prompts you to change the current element, if you desire. Enter the value for the current element, then press [R/S] to access the next element.	[A] [R/S] ⋮ [R/S] [R/S]	**1:1=**a_{11}**?** or **RE.1:1=**a_{11}**?** **1:2=**a_{12}**?** or **IM.1:1=**y_{11}**?** ⋮ n**:**n**=**a_{nn}**?** or **IM.**n**:**n**=**y_{nn}**?** **A I DT B SE**
To review and edit the matrix A, just repeat this process. To leave an entry unchanged, just press [R/S].		
5. To edit a specific element $a_{i,j}$, first enter the editor, then specify the element as *iii.jjj*.‡	[A] *iii.jjj* [A]	 i**:**j**=**$a_{i,j}$**?** or **RE.**i**:**j**=**$a_{i,j}$**?**
If *iii.jjj* does not exist, the editor ends and returns to the main menu.		
Use [R/S] to proceed to subsequent elements and finally exit the editor.	⋮ [R/S]	⋮ **A I DT B SE**

Instruction Table for MATRX (Continued)

Instructions	Key In:	Display
6. To *only view* the matrix A:	■ [A] [R/S] § ⋮ [R/S] §	**1:1**=a_{11} or **RE.1:1**=a_{11} **1:2**=a_{12} or **IM.1:1**=y_{11} ⋮ **A I DT B SE**
Note there is no **?** prompt. You cannot change these entries.		
7. To enter, edit, and view the column matrix, B, follow exactly steps 4, 5, or 6, but use [D] (**B**) and ■ [D] (■**B**). B is automatically correctly dimensioned to one column by *n* rows (step 3).		
8. To end the editor and return to the menu:	[J]	**A I DT B SE**
9. Execute a matrix operation:		
■ Invert A to A^{-1}. This replaces matrix A.	[B] (**I**)	**A I DT B SE**
View A^{-1}.	■ [A] [R/S] § ⋮	
■ Transpose A (if real) to A^T. This replaces matrix A. (If you had inverted A, be sure to re-invert it first. If you had found det(A) or solved for X, you must invert A *twice* to restore it before transposing it. Refer to "Remarks" for this section for more information.)	■ [B]	**A I DT B SE**
View A^T.	■ [A] [R/S] § ⋮	
■ Determinant of A (if real), det (A). (If you had inverted A, be sure to re-invert it first.) (This operation replaces A with its LU-decomposed form. See "Remarks".)	[C] (**DT**) [R/S] §	**DET**=*result* **A I DT B SE**

Instruction Table for MATRX (Continued)

Instructions	Key In:	Display
■ Solve the system of equations described by AX = B. This finds X, which replaces B. (It also replaces A with its LU-decomposed form. See "Remarks".) View X (replaces B).	[E] (SE) ■ [D] (■ B) [R/S] § ⋮	A I DT B SE

* The size of this program depends on the size of the matrices involved. It is ($order^2 + order + 2$) for real matrices **A** and **B**; [$4(order^2) + 2(order) + 2$] for complex matrices **A** and **B**. However, note that the program will tell you what memory size to set if it is not large enough.

† To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

‡ You can drop leading zeros in the *i*-part and trailing zeros in the *j*-part. A zero part defaults to a 1. For example, 0.000 defaults to 1.001.

§ If you have a printer attached, the display automatically returns to the main menu after printing the result(s).

For a list of error messages relevant to matrix operations, see "Error Messages" in the next chapter.

Remarks

Alteration of the Original Matrix. The input matrix **A** is altered by the operations finding the inverse, the determinant, the transpose, and the solution of the matrix equation. You can re-invert $\mathbf{A}^{-1}$ and re-transpose $\mathbf{A}^{T}$ to restore the original form of **A**. However, if you have calculated the determinant or the solution matrix, then **A** is in its *LU-decomposed form*. To restore **A**, simply *invert it twice*. The LU-decompostion does *not* interfere with any subsequent MATRX operation *except* transposition and editing*. For more information on LU-decomposition, refer to "LU-Decomposition" in the next chapter ("Matrix Functions").

* Do not attempt to edit an LU-decomposed matrix unless you intend to change every element.

Matrix Storage. The MATRX program stores a matrix **A** starting in R_0 of main memory; it is named **R0**. Its column matrix **B** is stored after it, and the result matrix **X** overwrites **B**. Refer to the chapter "Matrix Functions" for an explanation of how matrices are named and stored, and how much room they need.

MATRX cannot access any other matrices, with the exception of the previous **R0** and its corresponding column matrix.

Redefined Keys. This program uses local Alpha labels (as explained in the owner's manual for the HP-41) assigned to keys [A]–[E], [J], ■[A], ■[B], and ■[D]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

Examples

Given the system of equations at the beginning of this chapter, we have the matrix equation **AX** = **B**, or

$$\begin{bmatrix} 3.8 & 7.2 \\ 1.3 & -0.9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 16.5 \\ -22.1 \end{bmatrix}$$

Find the inverse, determinant, and transpose of **A**, and then find the solution matrix, **X**.

Keystrokes	Display	
[FIX] 4		Sets the display format used here.
[XEQ] [SIZE] 008		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 008.
[XEQ] [MATRX]	RL CX	Starts the MATRX program.
[A] (RL)	ORDER=?	Selects a real matrix.
2 [R/S]	A I DT B SE	Dimensions a 2 × 2 square matrix.

Keystrokes	Display	
A	1:1=a_{11}?	Enters the editor for **A** and displays (old) element a_{11}.
3.8 R/S	1:2=a_{12}?	Enters 3.8 for a_{11}.
7.2 R/S	2:1=a_{21}?	
1.3 R/S	2:2=a_{22}?	
.9 CHS R/S	A I DT B SE	Enters a_{22} and returns main menu.
A	1:1=3.8000	Displays the current contents of **A** for your review.
R/S *	1:2=7.2000	
R/S *	2:1=1.3000	
R/S *	2:2=−0.9000	
R/S *	A I DT B SE	
B (I)	A I DT B SE	Inverts **A**.
A	1:1=0.0704	Displays the current contents of **A**, now $\mathbf{A}^{-1}$.
R/S *	1:2=0.5634	
R/S *	2:1=0.1017	
R/S *	2:2=−0.2973	
R/S *	A I DT B SE	
B (I)	A I DT B SE	Reinverts $\mathbf{A}^{-1}$ to the original **A**.
B	A I DT B SE	Transposes **A**.
A	1:1=3.8000	Displays the current contents of **A**, now $\mathbf{A}^{T}$.
R/S *	1:2=1.3000	
R/S *	2:1=7.2000	
R/S *	2:2=−0.9000	
R/S *	A I DT B SE	
B	A I DT B SE	Retransposes $\mathbf{A}^{T}$ to the original **A**.
C (DT)	DET=−12.7800	Det(**A**).
D (B)	1:1=b_{11}?	Enters the editor for **B** and displays (old) element b_{11}.
16.5 R/S	2:1=b_{21}?	Enters 16.5 for b_{11}.
22.1 CHS R/S	A I DT B SE	Enters b_{21} and returns main menu.
D (B)	1:1=16.5000	Displays the current contents of **B** for your review.
R/S *	2:1=−22.1000	
R/S *	A I DT B SE	
E (SE)	A I DT B SE	Solves the system **AX** = **B**, placing **X** in **B**.

Keystrokes	Display	
■ [D] (■ B)	1:1=−11.2887	Displays the solution matrix (in **B**).
[R/S] *	2:1=8.2496	
[R/S] *	A I DT B SE	

Find the inverse of this complex matrix:

$$\begin{bmatrix} 1 + 2i & 3 + 3i \\ 4 + 5i & 6 + 7i \end{bmatrix}$$

Keystrokes	Display	
[XEQ] [SIZE] 017		For one complex matrix **A**, $2^2 \times 4 + 1 = 17$.
[XEQ] [MATRX]	RL CX	Starts the program over.
[B] (CX)	ORDER=?	
2 [R/S]	A I DT B SE	Dimensions a 2 × 2 complex matrix.
[A]	RE.1:1=a_{11}?	
1 [R/S]	IM.1:1=y_{11}?	
2 [R/S]	RE.1:2=a_{12}?	
3 [R/S]	IM.1:2=y_{12}?	
4 [R/S]	RE.2:1=a_{21}?	Oops! Wrong entry for y_{12}. Should be 3, not 4.
1.002 [A]	RE.1:2=3.0000?	Moves editor back to a_{12}.
[R/S]	IM.1:2=4.0000?	The imaginary part. (Wrong value.)
3 [R/S]	RE.2:1=a_{21}?	Correct value is entered for y_{12}. Proceed with data entry.

* [R/S] keystroke not necessary if a printer is attached.

Keystrokes	Display	
4 [R/S]	IM.2:1=y_{21}?	
5 [R/S]	RE.2:2=a_{22}?	
6 [R/S]	IM.2:2=y_{22}?	
7 [R/S]	A I DT B SE	Enters last element and returns main menu.
[B] (I)	A I DT B SE	Inverts **A**.
[■] [A]	RE.1:1=−0.9663	Viewing $\mathbf{A}^{-1}$.*
2.002 [A]	RE.2:2=−0.2360	Displays $a_{22} + iy_{22}$.*
[R/S]	IM.2:2=−0.0225	
[R/S] (or [J])	A I DT B SE	Exits the editor.

* If you have a printer attached, then the viewing operation automatically prints the entire matrix and redisplays the menu.

THE MATRIX FUNCTIONS

Contents

THE MATRIX FUNCTIONS

This chapter is a companion to the preceding chapter, "The Matrix Program." This chapter comprehensively covers all matrix functionality available in this pac for the advanced user.

You can create, manipulate, and store real and complex matrices. The size and number of matrices is limited only by the amount of memory available in the calculator. If you have extended memory (an HP 82180A Extended Functions/Memory Module or an HP-41CX), you can also store matrices there.

The matrix operations offered in this pac include inversion, transposition, finding the determinant, solving a system of equations, and doing matrix arithmetic. In addition, you can manipulate individual elements in and between matrices.

Setting Up a Matrix

To create a matrix you must provide its name and dimensions. The function [MATDIM] uses the name in the Alpha register and the dimensions *mmm.nnn* in the X-register to create a matrix.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & a_{ij} & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

It does *not* clear (zero) the elements of a new matrix in main memory, but retains the existing contents of the previous matrix or registers. It *does* clear the elements of a new matrix in extended memory.

You then enter values—numeric or Alpha—into a matrix via the matrix editor (page 34).

Naming a Matrix

The name you give a matrix determines where it will be stored. A matrix to be stored in main (non-extended) memory must be named

R*xxx*,

where *xxx* is up to three digits. (You can drop leading zeros.) The matrix will be stored *starting* in R_{xxx}. For example, **R007** is the same as **R7**, which would store this matrix header in R_{07}.

As a shortcut, if you specify matrix **R**, its name and location will be **R0**.

A matrix to be stored in extended memory can be named with up to seven Alpha characters, excepting just the letter "X" (which is reserved to name the X-register) and the letter "R" followed by up to three digits (which is reserved to name the main memory arrays). You do not need to specify a file type; it will automatically be given one unique to matrices.

Use the Alpha register to specify matrix names. When specifying more than one name (as parameters for certain functions), separate them with commas.

MNAME? returns the name of the current matrix to the Alpha register.

Dimensioning a Matrix

Specify the dimensions of a new matrix as *mmm.nnn*, where *m* is the number of rows and *n* is the number of columns. You can drop leading zeros for *m* and trailing zeros for *n*.

For a complex matrix, specify *mmm.nnn* as *twice* the number of rows and *twice* the number of columns. (Refer to "Working with Complex Matrices.")

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1+i & 2+3i \\ 4+5i & 6+7i \end{bmatrix}$$

$$mmm.nnn = 2.003 \qquad mmm.nnn = 4.004$$

A zero part defaults to a 1, so 0 is equivalent to 1.001, 3 to 3.001, and .023 to 1.023.

[MATDIM] dimensions a new matrix or redimensions an existing one to the given dimensions.

[DIM?] returns the dimensions *mmm.nnn* of the matrix specified in the Alpha register to the X-register. (A blank Alpha register specifies the current matrix.)

How a Matrix Is Stored

The elements of a matrix are stored in memory in order from left to right along each row, from the first row to the last. Each element occupies one data-storage register. A complex number requires four registers to store its parts.

Memory Space. A matrix in main memory occupies $(m \times n) + 1$ data-storage registers, one register being used as a status header. A complex matrix uses $(2m \times 2n) + 1$ registers, where m is the number of rows in the complex matrix and n is the number of columns in the complex matrix.

> **Note:** To successfully dimension a matrix in main memory, the size of data-storage memory must be large enough to hold it. If it is *not*, you will see the message **NONEXISTENT** when you try [MATDIM]. Reallocate more registers to data storage ([SIZE] *nnn*) and try again.

A matrix in extended memory has a file length of $m \times n$. ($2m \times 2n$ for a complex matrix.) Its file type is unique to matrices. Do not use the function [CLFL] with a matrix in extended memory: this destroys part of the file's header information. Instead, use [PURFL] to purge the entire matrix.

Changing Matrix Dimensions. If you redimension a matrix to a different size, then the existing elements are reassigned to new elements according to the new dimensions. Extra old elements are lost; extra new elements take on the values already present in the new registers—except in extended memory, where new elements are set to zero.

CAUTION

When [MATDIM] is used to redimension a matrix stored in extended memory, the position of the matrix pointer is not readjusted. If the pointer happened to be positioned to an element that is outside the new bounds of the redimensioned matrix, it must be repositioned to be within the new bounds by executing either [MSIJ] or [MSIJA] with valid indices before the pointer can be used again.

Existing matrices in extended memory cannot be redimensioned to completely fill extended memory. The maximum allowable size of a redimensioned matrix is one register less than the currently available extended memory. A *new* matrix can, however, be dimensioned to completely fill available extended memory.

Redimensioning 2 × 3 to 2 × 2

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \begin{matrix} \text{lost} \\ 5 \quad 6 \end{matrix}$$

Redimensioning 2 × 3 to 2 × 4

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & ? & ? \end{bmatrix}$$

This is what happens each time you dimension a new matrix since the old elements from the previous current matrix remain until you change them.

Using the Matrix Editors

There are two matrix editors: [MEDIT] for real matrices and [CMEDIT] for complex matrices. They are otherwise quite similar.

The matrix editors are used for three purposes:

- Entering new values into the elements of a matrix.
- Reviewing and changing ("editing") the elements of a matrix, either in order or by "random access" of specific elements.
- Viewing (without being able to change) the elements of a matrix (flag 08 set).

When you execute [MEDIT] or [CMEDIT], the editor displays element 1,1 of the matrix specified in the Alpha register or of the current matrix if the Alpha register is empty. Pressing [R/S] steps the display through the elements; for a complex matrix, each part of the complex element is shown separately.

Function	Display	Function	Display
[MEDIT]	1:1=1.0000?	[CMEDIT]	RE.1:1=1.0000?
[R/S]	1:2=2.0000?	[R/S]	IM.1:1=1.0000?
		[R/S]	RE.1:2=2.0000?
⋮	⋮	⋮	⋮
[R/S]	*(X-register)*	[R/S]	*(X-register)*

The **?** at the end of the display line indicates that you can change that value. In effect, you are being asked whether this is the value you want. If you want to change the element you see, just enter the new value and press [R/S]. You do this for a brand new matrix as well as for correcting or altering a single value.

If you press [R/S] without entering a new value, the current value remains unchanged.

Viewing without Editing. If you set flag 08, the editor will let you only view the elements, not change them. The display appears without the **?** at the end of the line.

1:1=1.0000

If you have a printer attached while flag 08 is set, it will print out all the elements of the matrix without pausing.

Directly Accessing Any Element. You can directly access any specific element *while the editor is active* (and the User keyboard is also active). To access the element in the *i*th row and the *j*th column, enter *iii.jjj* and press [A]. (This is as in the MATRX program.) You can drop leading zeros in *iii* and trailing zeros in *jjj*.

For a complex matrix, you can directly access the *real part* of element *i*, *j*. Then use [R/S] to access its imaginary part.

Keystrokes	**Display**
[ALPHA] *matrix name* [ALPHA]	
[XEQ] [MEDIT]	1:1=1.0000?
3.003 [A]	3:3=6.0000?
[ALPHA] *complex-matrix name* [ALPHA]	
[XEQ] [CMEDIT]	RE.1:1=1.0000?
3.003 [A]	RE.3:3=6.0000?
[R/S]	IM.3:3=7.0000?

You can drop leading zeros in the *i*-part and trailing zeros in the *j*-part. A zero part defaults to a 1.

Exiting the Editor. To leave the editor before it has reached the last element, do either:

- Press [J].
- Try to access a nonexistent element. For instance, in a 4 × 4 matrix, press 5 [A].

How to Specify a Matrix

Given the matrix multiplication operation **AB** = **C**, you know **A** and **B** and are looking for the product matrix, **C**. In performing this operation, the calculator must be given the identities of the existing matrices **A** and **B**, and also be told where to put the result matrix, **C**. (However, the result matrix can be the same as one of the input matrices.) *All* given matrices must already exist as named, dimensioned matrices. Naturally, only **A** and **B** must contain valid data.

Some functions use only one input matrix, and some functions *automatically* use one of the input matrices for output. So the minimum number of matrices to specify is one, and the maximum is three.

A matrix function checks the Alpha register for the names (that is, the locations) of the matrices it needs for input and output. Before executing that function, you should *specify* all needed parameters on one line in the Alpha register, separating each with a comma:

Alpha Register	*input matrix*[,*input matrix*][,*result matrix*]

For instance,

Alpha Register	A,B,C

XEQ M*M

will multiply the matrices **A** and **B**, putting the result in existing matrix **C**.

Scalar Operations. Scalar input and output must be in the X-register, and so this location does not need to be specified *unless* the function in question can use *either* a scalar *or* a matrix for the same input parameter. To specify the X-register, use X.

For instance, MATDIM *requires* a scalar input and a matrix name, so you do not need to specify the X-register. On the other hand, the scalar arithmentic functions, such as MAT*, can use *either* two matrices *or* a scalar and a matrix for input. Therefore, you must specify X if you want to use it.

The Current Matrix. The *current matrix* is the last one accessed (used) by a matrix operation. If the Alpha register is clear and you execute a matrix function that requires a matrix specification, the current matrix is used *by default*. (If there is no current matrix, **UNDEF ARRAY** results.)

The result matrix of a matrix function becomes the current matrix following that operation.

To find out the name of the current matrix, execute MNAME?. Its name is returned into the Alpha register.

Default Matrix Parameters

If you *don't* specify any or all the matrices that a matrix function needs, then certain *default* parameters exist. (Default parameters are those automatically assumed if you don't specify them.) The most common default you will probably use is the current matrix. If you don't specify a particular matrix name *and the Alpha register is clear*, then the default matrix is the current one.

For matrix operations requiring up to three matrix names in the Alpha register, the following table gives the conventions to interpret the parameters.

Matrix Specifications

Alpha Register's Contents	Matrices Specified
A,B,C	A, B, C
A,B	A, B, B
A	A, A, A
A,,B	A, A, B
,A,B	current, A, B
,A	current, A, A
,,A	current, current, A
X,A,B	X-register, A, B
X,A	X-register, A, A
A,X	A, X-register, A
A,,X	A, A, A (ignores X)
X	X-register, current, current
(blank)	current, current, current

Error Messages

Refer to your HP-41 owner's documentation for error messages you don't see here.

ALPHA DATA results if the specified matrix contains Alpha data and so cannot be operated upon. The matrix is unchanged.

DATA ERROR results if the value in the X-, Y-, or Z-register is invalid.

DATA ERROR X results if the value in the X-register is invalid.

DATA ERROR Y results if the value in the Y-register is invalid.

DIM ERROR results if the dimension of the specified matrix is not correct for the current operation.

END OF ARRAY results if you attempt a function that uses the matrix pointer and the pointer is beyond its defined bounds.

NAME ERROR results if an invalid matrix name is specified (such as "X") or if the number of distinct matrix names is incorrect for a function.

NO ROOM results if there is not enough room to store a matrix in extended memory.

NO X-MEMORY results if you attempt to create a matrix in extended memory when your calculator has no extended memory (that is, an HP-41C/CV without an HP 82180A Extended Functions/Memory Module).

NONEXISTENT results if there are not enough storage registers in main memory to store the matrix. Re-size memory ([SIZE] *nnn*) to a larger figure to accommodate the new matrix.

NOT ARRAY FL results if you attempt a matrix operation on an extended-memory file that is not a matrix file.

NOT CPX results if you try to use [CMEDIT] with a real matrix of odd order.

TRY AGAIN results if you execute [MATDIM] with less than two available registers of program memory. Either resize data-storage memory to *fewer* data registers, or use [CLP] to eliminate a program.

UNDEF ARRAY results if you execute a function needing a matrix specification but the Alpha register does not contain a valid matrix specification.

Storing and Recalling Individual Matrix Elements

The matrix editor provides a method of storing and reviewing matrix elements. For programming, you can use the following functions to manipulate individual matrix elements.

A specific element is identified by the value *iii.jjj* for its location in the *i*th row of the *j*th column. You can drop leading zeros in the *i*-index and trailing zeros in the *j*-index.

Accessing Elements One by One

To store or recall an individual element, you first set or recall the element (row and column) pointer value *iii.jjj*, then store or recall the value of the element from or to the X-register. To go on to another element, you then either increment the pointer or reset it.

The value of the pointer defines the *current element*.

Setting and Recalling the Pointer

Function	Effect
MSIJA (*set pointer by Alpha*)	Sets element pointer of specified matrix to *iii.jjj*. **Input:** *matrix name* in Alpha reg. *iii.jjj* in X-reg.
MSIJ (*set pointer*)	Sets element pointer of current matrix to *iii.jjj*. **Input:** *iii.jjj* in X-reg.
MRIJA (*recall pointer by Alpha*)	Recalls element pointer of specified matrix to X-reg. **Input:** *matrix name* in Alpha reg. **Output:** *iii.jjj* into X-reg.
MRIJ (*recall pointer*)	Recalls element pointer of current matrix to X-reg. **Output:** *iii.jjj* into X-reg.

The following functions increment and decrement the element pointer rowwise (*iii*) or columnwise (*jjj*). If the end of a column is reached (with the *i*-index) or the end of a row is reached (with the *j*-index), then the index advances to the next larger or smaller column or row and sets flag 09. If the index advances beyond the size of the matrix, both flags 09 and 10 are set. These functions always either set or clear flags 09 and 10. If the conditions listed above don't occur, the flags are cleared every time the functions are executed.

Incrementing and Decrementing the Pointer

Function	Effect
I+	Increments *iii* of pointer by one.
I−	Decrements *iii* by one.
J+	Increments *jjj* of pointer by one.
J−	Decrements *jjj* by one.

Storing and Recalling the Element's Value

Function	Effect
MS (*matrix store*)	Stores value from X-reg into current element of current matrix. **Input:** value in X-reg.
MR (*matrix recall*)	Recalls value in current element of current matrix into X-reg. **Output:** value into X-reg.

Accessing Elements Sequentially

The following functions provide a faster, more automated alternative to adjusting the pointer value to access each element. These combine storing or recalling values and then incrementing or decrementing the *i*- or *j*-index, so that the pointer is automatically set to the next element.

Storing and Recalling the Element's Value

Function	Effect
MSC+ (*matrix store by column*)	Stores value from X-reg into current element and then advances pointer to next element in column. **Input:** value in X-reg.
MSR+ (*matrix store by row*)	Stores value from X-reg into current element and then advances pointer to next element in row. **Input:** value in X-reg.
MRC+ (*matrix recall by column*)	Recalls value to X-reg from current element and then advances pointer to next element in column. **Output:** value into X-reg.
MRR+ (*matrix recall by row*)	Recalls value to X-reg from current element and then advances pointer to next element in row. **Output:** value into X-reg.
MRC− (*matrix recall backwards by column*)	Recalls value to X-reg from current element and then decrements pointer to previous element in column. **Output:** value into X-reg.
MRR− (*matrix recall backwards by row*)	Recalls value to X-reg from current element and then decrements pointer to previous element in row. **Output:** value into X-reg.

When the end of a column or row is reached, the pointer's index advances to the next (or previous) column or row. If the pointer's index is moved beyond the boundaries of the matrix, it cannot be moved back using these functions. You must use MSIJ or MSIJA.

The following sequence of keystrokes will create the matrix **ABC** (in extended memory).

$$\mathbf{ABC} = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$$

Keystrokes	Display	
FIX 4		Sets the display format used here.
ALPHA ABC ALPHA		Matrix name in extended memory.
2.003 XEQ MATDIM	2.0030	Dimensions matrix **ABC** to 2 rows × 3 columns.
0 MSIJ	0.0000	Sets element pointer to 1.001.
5 MSR+	5.0000	Enters element and advances pointer to next column for next entry, setting flag 09.
6 MSR+	6.0000	
7 MSR+	7.0000	Pointer automatically moves to the second row.
8 MSR+	8.0000	
9 MSR+	9.0000	
10 MS	10.0000	
SF 08		This sets the editor to display only; if you have a printer attached this is a faster way to view the matrix elements.
XEQ MEDIT	1:1=5.0000 1:2=6.0000 1:3=7.0000 2:1=8.0000 2:2=9.0000 2:3=10.0000	Now let's look at **ABC**. (ABC is still in the Alpha register.) If you have no printer attached, press R/S to view each successive element.
		Exits editor.

Matrix Functions

This section briefly defines the matrix functions besides the dimensioning, storing, and recalling functions discussed above. On page 58 is a Function Table that lists all matrix functions in this pac.

Note that most of these functions are not meaningful for matrices containing Alpha data and that many of these functions are not meaningful for complex matrices. In any case, a complex matrix appears as a real matrix to all functions except [CMEDIT]. Refer to "Working with Complex Matrices" for more information on using these functions with complex matrices.

Matrix Arithmetic

The matrix-arithmetic functions provided are scalar addition, subtraction, multiplication, and division, as well as true matrix multiplication. The scalar arithmetic functions can use two matrices as operands, or one scalar and one matrix. When using two matrices, the matrices do not have to be of the same dimension, but the total number of elements in each must be the same. This also applies to the result matrix. (Note that the *i-j* notation in the table below assumes that the dimensions of the matrices are the same. If this is not the case, the *i-j* notation does not apply.)

Matrix multiplication, on the other hand, calculates each new element by summing the products of the first matrix's row elements by the second's column elements. The number of columns in the first matrix must equal the number of rows in the second matrix. The result matrix must have the same number of rows as the first matrix and the same number of columns as the second matrix.

If there is a scalar operand, it must be in the X-register, and X must be specified in the Alpha register.

Function	Effect
	Scalar Arithmetic
[MAT+] (*matrix add*)	Adds a scalar or matrix element to each element. **Input:** *matrix name A* or **X**,*matrix name B* or **X**, *result-matrix name C* in Alpha reg. **Output:** $c_{ij} = a_{ij} + x$ *or* $c_{ij} = x + b_{ij}$ *or* $c_{ij} = a_{ij} + b_{ij}$ for all *i*, *j* in *C*.
[MAT−] (*matrix subtract*)	Subtracts a scalar or matrix element from each element. **Input:** *matrix name A* or **X**,*matrix name B* or **X**, *result-matrix name C* in Alpha reg. **Output:** $c_{ij} = a_{ij} - x$ or $c_{ij} = x - b_{ij}$ or $c_{ij} = a_{ij} - b_{ij}$ for all *i*, *j* in *C*.

Function	Effect
MAT* (*scalar matrix-multiply*)	Multiplies a scalar or matrix element by each element. **Input:** *matrix name A* or **X,***matrix name B* or **X,** *result-matrix name C* in Alpha reg. **Output:** $c_{ij} = a_{ij} \times x$ or $c_{ij} = x + b_{ij}$ or $c_{ij} = a_{ij} \times b_{ij}$ for all i, j in C.
MAT/ (*matrix divide*)	Divides a scalar or matrix element into each element. **Input:** *matrix name A* or **X,***matrix name B* or **X,** *result-matrix name C* in Alpha reg. **Output:** $c_{ij} = a_{ij} \div x$ or $c_{ij} = x \div b_{ij}$ or $c_{ij} = a_{ij} \div b_{ij}$ for all i, j in C.
	Nonscalar Arithmetic
M*M (*matrix multiplication*)	Calculates each new element i, j by multiplying the ith row in A by the jth column in B. **Input:** *matrix name A*,*matrix name B*,*result-matrix name C* in Alpha reg., where C must be different from A and B. **Output:** $c_{ij} = \sum_{k=1}^{p} a_{ik} \times b_{kj}$, where A has p columns and B has p rows.

Major Matrix Operations

The major matrix operations are: inversion, finding the determinant, transposition, and solving a system of linear equations.

A system of linear equations

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

can be represented by the matrix equation $\mathbf{AX} = \mathbf{B}$, where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \text{and } \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

A is the *coefficient matrix*, **B** is the *constant* or *column matrix*, and **X** is the *solution matrix*. (The **B** matrix is overwritten by the **X** matrix when solving this system.)

Function	Effect
MDET (*determinant*)	Finds the determinant of the given real square matrix. **Input:** *matrix name* in Alpha reg. **Output:** determinant into X-reg. (Replaces matrix with LU-decomposed form).
MINV (*inverse*)	Inverts and replaces the given square matrix. **Input:** *matrix name* in Alpha reg. **Output:** Replaces matrix with its inverse.
MSYS (*system of equations*)	Solves a system of linear equations. **Input:** *matrix name A*,*matrix name B* in Alpha reg. **Output:** solution matrix *X* replaces *B* in the system defined by the matrix equation $AX = B$. (Replaces *A* with its LU-decomposed form.)
TRNPS (*transpose*)	Transposes and replaces the given real matrix. **Input:** *matrix name* in Alpha reg. **Output:** Replaces matrix with its transpose.

Note: You cannot transpose or change any element of a matrix *A* that has had its determinant found or has had its solution matrix found because MDET and MSYS transform the input matrix *A* into its *LU-decomposed form*. (Refer to "LU-Decomposition" for more information.) However, you *can* retrieve the original form of *A* from its decomposed form by inverting it *twice* (execute MINV twice). The LU-decomposition does *not* interfere with the calculations for MINV, MSYS, or MDET.

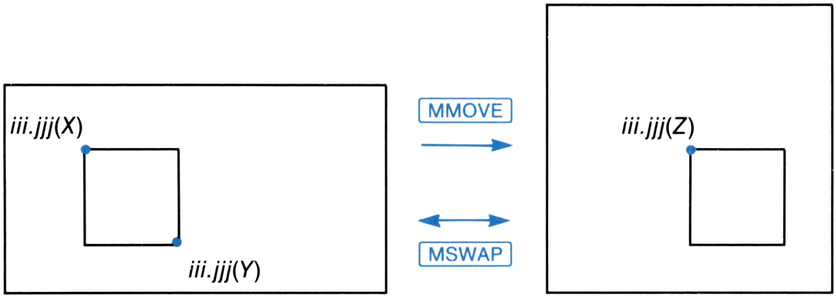
Other Matrix Functions ("Utilities")

The remaining matrix functions, also called *utilities*, are those for copying and exchanging parts of matrices, and miscellaneous, extra arithmetic functions: finding sums, norms, maxima, and minima, and matrix reduction.

Moving and Exchanging Matrix Sections

Function	Effect
C< >C (*exchange columns*)	Exchanges columns *k* and *l* in a matrix. **Input:** *matrix name* in Alpha reg. *kkk.lll* in X-reg.
R< >R (*exchange rows*)	Exchanges rows *k* and *l* in a matrix. **Input:** *matrix name* in Alpha reg. *kkk.lll* in X-reg.
MMOVE (*matrix move*)	Copies the submatrix defined by pointers in source matrix to the area defined by one pointer in target matrix. **Input:** *source-matrix name A*,*target-matrix name B* in Alpha reg. in X-reg: *iii.jjj* for *A*'s initial element; in Y-reg: *iii.jjj* for *A*'s final element; in Z-reg: *iii.jjj* for *B*'s initial element.
MSWAP (*matrix swap*)	Exchanges the submatrix defined by pointers in a source matrix with the area defined by one pointer in a target matrix. **Input:** *matrix name A*,*matrix name B* in Alpha reg. in X-reg: *iii.jjj* for *A*'s initial element; in Y-reg: *iii.jjj* for *A*'s final element; in Z-reg: *iii.jjj* for *B*'s initial element.

When executing MMOVE and MSWAP, if *A* and *B* are the same matrix and the source submatrix overlaps the target submatrix, the elements are processed in the following order: reverse column order (last to first) and reverse element order (last to first) within each column.



Source matrix (*A*) Target matrix (*B*)

When an input of the form *iii.jjj* is expected in the X-register, a zero value for either the *i*-part or the *j*-part is interpreted as 1. (Zero alone equals 1.001.) This is true for the *iii.jjj*-values that MMOVE and MSWAP expect in the X- and Z-registers, *but not for the pointer value in the Y-register.*

For the Y-register input, a zero value for the *i*-part is interpreted as *m*, the last row, while a zero value for the *j*-part is interpreted as *n*, the last column. For example, in a 4 × 5 matrix,

Y-Register	Pointer Value
0.000	4.005
3.000	3.005
0.003	4.003

This convention facilitates easy copying (or exchanging) of entire matrices because simply by clearing the stack (CLST) or entering three zeros you specify the elements 1.001 (X) and *mmm.nnn* (Y) for the first matrix and element 1.001 (Z) for the second matrix, thus defining two entire matrices.

The following instructions would copy the matrix **R0** of unspecified dimensions to a new matrix **R30**:

ALPHA R0 ALPHA
XEQ DIM?
ALPHA R30 ALPHA
XEQ MATDIM
ALPHA R0,R30 ALPHA
XEQ CLST
XEQ MMOVE

Miscellaneous Arithmetic Functions

Function	Effect
	Maxima and Minima
MAX (*maximum*)	Finds maximum element in matrix. Sets element pointer to it. **Input:** *matrix name* in Alpha reg. **Output:** maximum value into X-reg.
MIN (*minimum*)	Finds minimum element in matrix. Sets element pointer to it. **Input:** *matrix name* in Alpha reg. **Output:** minimum value into X-reg.
MAXAB (*maximum absolute value*)	Finds maximum absolute value in matrix. Sets element pointer to it. **Input:** *matrix name* in Alpha reg. **Output:** maximum absolute value into X-reg.
CMAXAB (*column's maximum absolute value*)	Finds maximum absolute value in *k*th column. Sets element pointer to it. **Input:** *matrix name* in Alpha reg. *kkk* in X-reg. **Output:** maximum absolute value into X-reg.
RMAXAB (*row's maximum absolute value*)	Finds maximum absolute value in *k*th row. Sets element pointer to it. **Input:** *matrix name* in Alpha reg. *kkk* in X-reg. **Output:** maximum absolute value into X-reg.
	Norms
CNRM (*column norm*)	Finds the largest sum of the absolute values of the elements in each column of matrix. Sets element pointer to first element of column with largest sum. **Input:** *matrix name* in Alpha reg. **Output:** column norm into X-reg.
FNRM (*Frobenius norm*)	Finds the square root of the sum of the squares of all elements in matrix. **Input:** *matrix name* in Alpha reg. **Output:** Frobenius norm into X-reg.
RNRM (*row norm*)	Finds the largest sum of the absolute values of the elements in each row of matrix. Sets element pointer to first element of row with largest sum. **Input:** *matrix name* in Alpha reg. **Output:** row norm into X-reg.

Miscellaneous Arithmetic Functions (Continued)

Function	Effect
	Sums
SUM	Sums all elements in matrix. **Input:** *matrix name* in Alpha reg. **Output:** sum in X-reg.
SUMAB (*sum of absolute values*)	Sums absolute values of all elements in matrix. **Input:** *matrix name* in Alpha reg. **Output:** sum of absolute values in X-reg.
CSUM (*column sum*)	Finds the sum of each column and stores them in result vector. **Input:** *matrix name,result-matrix name* in Alpha reg. Number of elements in result matrix must equal number of columns in input matrix.
RSUM (*row sum*)	Finds the sum of each row and stores sums in result vector. **Input:** *matrix name,result-matrix name* in Alpha reg. Number of elements in result matrix must equal number of rows in input matrix.
	Other
YC+C (*Y times column plus column*)	Multiplies each element in column k of matrix by value in Y-reg. and adds it to corresponding element in column l, thereby changing the elements in column l. That is, converts a_{il} to $a_{il} + y \times a_{ik}$. **Input:** *matrix name* in Alpha reg. *kkk.lll* in X-reg. y in Y-reg.
PIV (*pivot*)	Finds the pivot value in column k; that is, the maximum absolute value of an element on or below the diagonal. **Input:** *matrix name* in Alpha reg. *kkk* in X-reg. **Output:** pivot value in X-reg.; pointer set to pivot element.
R>R? (*compare rows*)	Compares elements in rows k and l. If (and only if) the first non-equal element in k is greater than its corresponding element in l, then the comparison is positive for the "do if true" rule of programming. **Input:** *matrix name* in Alpha reg. *kkk.lll* in X-reg. **Output: YES** if first non-equal element in row k is greater than element in row l. **NO** in all other cases.

Miscellaneous Arithmetic Functions (Continued)

Function	Effect
AIP (*Alpha recall of integer part*)	Appends the integer part of the number in the X-register to the contents of the Alpha register. For $x < 0$, AIP appends the absolute value.
MP (*Alpha recall of matrix prompt*)	Appends a matrix prompt *rrr.ccc=* to the contents of the Alpha register.

Working with Complex Matrices

When working with complex matrices it is most important to remember that, in the calculator, a complex matrix is simply a real matrix with four times as many elements. Only the MATRX program and the complex-matrix editor (CMEDIT) "recognize" a matrix as complex and treat its elements accordingly. All other functions treat the real and imaginary parts of the complex elements as *separate* real elements.

How Complex Elements are Represented

In its internal representation a complex matrix has twice as many columns *and* twice as many rows as it "normally" would.

The complex number $100 + 200i$ is stored as

$$\begin{bmatrix} 100 & -200 \\ 200 & 100 \end{bmatrix}$$

The 2×1 complex matrix

$$\begin{bmatrix} 1 + 2i \\ 3 - 4i \end{bmatrix} \text{ is stored as } \begin{bmatrix} 1 & -2 \\ 2 & 1 \\ 3 & 4 \\ -4 & 3 \end{bmatrix}$$

There is one important exception to this scheme: for the column matrix (a vector) in a system of simultaneous equations.

Solving Complex Simultaneous Equations. The easiest way to work with complex matrices is to use the MATRX program. It automatically dimensions input and output complex matrices. However, MSYS can solve more complicated systems of equations than MATRX can.

In addition, a complex result-matrix from the MATRX program cannot be used for many complex-matrix operations *outside* of MATRX. This is because MATRX will dimension a complex *column* matrix differently than $2m \times 2$. *Instead*, it uses the dimensions $2m \times 1$, in which the real and imaginary parts of a number become successive elements in a single column.

This form has the advantage of saving memory and speeding up operations. The complex-matrix editor and [MSYS] can also use this $2m \times 1$ form, though they do not require it. This means you can use [MSYS] on a matrix system from MATRX.

You can convert an existing $2m \times 2$ complex column matrix to the $2m \times 1$ form by transposing it, redimensioning it to $1 \times 2m$, then re-transposing it. There is no easy way back.

Accessing Complex Elements. If you use the complex-matrix editor ([CMEDIT] or the editor in the MATRX program), you can access complex elements as if they were actual complex numbers. Otherwise (such as when you use pointer-setting functions), you must access complex elements as real elements stored according to the $2m \times 2n$ scheme given above.

Storage Space in Memory. Since the dimensions required for a complex matrix are four times greater than the actual number of complex elements (an $m \times n$ complex matrix being dimensioned as $2m \times 2n$), realize that the number of registers a complex matrix occupies in memory is correspondingly four times greater than a real matrix with the same number of elements. In other words, think of a complex matrix's storage size in terms of its [MATDIM] or [DIM?] dimensions, not its number of complex elements.

Using Functions with Complex Matrices

Most matrix functions do not operate meaningfully on complex matrices: since they don't recognize the different parts of a complex number as a single number, the results returned are not what you would expect for complex entries.

Valid Complex Operations. Certain matrix functions work equally well with real and complex functions. These are:

- [MSYS] Solving simultaneous equations
- [MINV] Matrix inverse
- [MAT+] Matrix add
- [MAT−] Matrix subtract
- [MAT*] Matrix scalar multiply, *but only by a real scalar in X-reg.*
- [M*M] Matrix multiplication

Both the input and result matrices must be complex.

LU-Decomposition

The *lower-upper (LU) decomposition* is an unrecognizably altered form of a matrix, often containing Alpha data. This transformation properly occurs in the process of finding the:

- Solution to a system of equations ([MSYS]; **SE** in the MATRX program).
- Determinant ([MDET]; **DT** in MATRX program).
- Inverse ([MINV]; **I** in MATRX program).

The first two of these operations convert the input matrix to its LU-decomposed form *and leave it there,* whereas inversion leaves the matrix in its inverted form. When you use functions that produce an LU-decomposed form, there are several things that you need to be aware of:

- You cannot edit an LU-decomposed matrix unless you edit every element (refer to "Editing and Viewing an LU-Decomposed Matrix," below for more details).
- You cannot perform any operation that will modify the matrix (other than [MINV]) because the LU status of the matrix will be cleared and it will become unrecognizable. Operations that have this effect are: [R<>R], [C<>C], [MS], [MSR+], [MSC−], [MSC+], [MMOVE] (intramatrix), [MSWAP], and [TRNPS].
- Care must be exercised when *viewing* an LU-decomposed matrix. Certain operations can alter elements without your knowledge (refer to "Editing and Viewing an LU-Decomposed Matrix," below, for more details).
- LU-decomposition destroys the original form of the matrix. So if you perform [MSYS] or [MDET] and then try to look at your input matrix (**A** in the MATRX program), *you will find only the altered, decomposed form.*
- You cannot calculate the transpose ([TRNPS]; ■ [B] in MATRX program) of a matrix in LU-decomposed form. LU-decomposition *does not hinder the correct calculation of the inverse, determinant, or solution matrix,* since these operations require the LU-decomposition anyway.

Reversing the LU-Decomposition. To restore a matrix to its original form from its decomposed form, simply *invert it twice* (in effect: find the inverse and then re-invert to the original). Naturally, for this to work the matrix must be invertible (non-singular). The result can differ slightly from the original due to rounding-off during operations.

Editing and Viewing an LU-Decomposed Matrix. LU-decomposed matrices are stored in a different form than normal matrices:

- Certain elements contain alpha data.
- The matrix status register is modified to indicate that the matrix is in LU form.

Editing *any* element of the matrix will clear the LU-flag in the status register, which makes the matrix unrecognizable to the program. Because of this, if you edit one element, you must edit them all if you wish to use the matrix again. Note that the matrix will no longer be in LU-decomposed form after this action.

You can *view* the contents of an LU-decomposed matrix by doing one of the following:

- From the main menu press ■ [A] (View A) to view individual elements without modifying them.
- Set flag 08 before executing [MEDIT] or [CMEDIT]. This allows you to view the elements without modifying them.

Examples

Find the determinant of the inverse of the transpose of the matrix below.

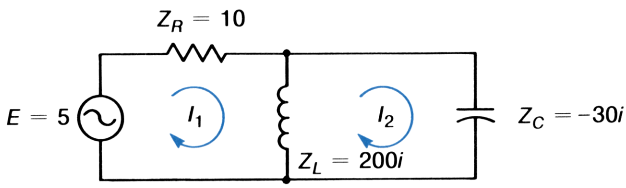
$$\begin{bmatrix} 6 & 3 & -2 \\ 1 & 4 & -3 \\ 2 & 3 & -1 \end{bmatrix}$$

The size of data-storage memory must be at least 10 registers ([SIZE] 010).

Keystrokes	Display	
FIX 4		Sets the display format used here.
SIZE 010		
ALPHA R0 ALPHA		Names matrix **R0** to be stored in main memory from R_{00}–R_{10}.
3.003 XEQ MATDIM	3.0030	Dimensions **R0** to 3×3.
CF 08		Sets editor to allow editing.
XEQ MEDIT	1:1= ?	The matrix editor prompts you for new elements, showing you old elements or the previous contents of the registers.
6 R/S	1:2= ?	
3 R/S	1:3= ?	
2 CHS R/S	2:1= ?	
1 R/S	2:2= ?	
4 R/S	2:3= ?	
3 CHS R/S	3:1= ?	
2 R/S	3:2= ?	
3 R/S	3:3= ?	
1 CHS R/S		Exits editor.
XEQ TRNPS		**R0** is transposed.
XEQ MINV		**R0** (which was transposed) is inverted.
XEQ MDET	0.0400	The determinant of the inverse of the transpose of the original matrix.

Note that if you had wanted to find the transpose of the *original* matrix *after* having found its determinant, you would have needed to invert the matrix twice to change the LU-decomposed form back to the original matrix.

Find the currents I_1 and I_2 in the electrical circuit shown below. The impedances of the components are indicated in complex form.



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

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

The size of data-storage memory must be set to at least 26 registers (SIZE 026) to accommodate two complex matrices.

Keystrokes	Display	
ALPHA R ALPHA 4.004 MATDIM	4.0040	Dimensions the complex coefficient matrix **R0** to 4 × 4 for its 2 rows and 2 columns. It needs 17 registers.
CMEDIT	RE.1:1= ?	Complex-matrix editor.
10 R/S 200 R/S	RE.1:2= ?	Loads the real and imaginary parts of elements into **R0**, the coefficient matrix (A).
0 R/S 200 CHS R/S	RE.2:1= ?	
0 R/S 200 CHS R/S	RE.2:2= ?	
0 R/S 170 R/S	−170.0000	
ALPHA R17 ALPHA 4.002 MATDIM	4.0020	Dimensions the column matrix **R17** to 4 × 2 for 2 complex rows and 1 complex column. It needs 9 registers.
CMEDIT	RE.1:1= ?	Complex-matrix editor.
5 R/S 0 R/S	RE.2:1= ?	Loads the real and imaginary parts of elements into **R17**, the column matrix (B).
0 R/S 0 R/S	0.0000	

Keystrokes	Display	
ALPHA R,R17 ALPHA XEQ MSYS	0.0000	Calculates the solution matrix (X) and loads it into **R17**.
SF 08		Sets editor for view-only operation.
ALPHA R17 ALPHA XEQ CMEDIT	RE.1:1=0.0372	Displays the complex results for I_1 and I_2, which are in **R17**. If you have a printer attached and set flag 08 before executing CMEDIT, all elements will be printed out automatically.
R/S	IM.1:1=0.1311	
R/S	RE.2:1=0.0437	
R/S	IM.2:1=0.1543	

The solution is

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

This last example asks you to solve a set of two simultaneous equations with two unknown variables. This requires the use of MSYS.

Silas Farmer has the following record of sales of cabbage and broccoli for three different weeks. He knows the total weight of produce sold each week, the total price received each week, and the price per pound of each crop. Determine the weights of cabbage and broccoli he sold each week.

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

The price of cabbage is $0.24/kg and the price of broccoli is $0.86/kg.

The following set of linear equations describes the two unknowns (the weights of cabbage and broccoli) for all three weeks, where the first row of the constant matrix represents the weights of cabbage for the three weeks and the second row represents the weights of broccoli. Since the constant matrix is not a column matrix, you must use MSYS and not the **SE** function in the **MATRX** program.

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

The size of data-storage memory must be set to at least 12 registers ([SIZE] 012) to accommodate these two real matrices.

Keystrokes	Display	
[ALPHA] R [ALPHA] 2.002 [XEQ] [MATDIM]	**2.0020**	Dimensions the coefficient matrix **R0** to 2 × 2.
[ALPHA] R5 [ALPHA] 2.003 [XEQ] [MATDIM]	**2.0030**	Dimensions the constant matrix **R5** to 2 × 3.
[CF] 08		Set editor to allow editing.
[XEQ] [MEDIT]	**1:1= ?**	Calls the matrix editor for the current matrix, which is **R5**.
274 [R/S] 233 [R/S] 331 [R/S] 120.32 [R/S] 112.96 [R/S] 151.36 [R/S]	**1:3= ?** **2:2= ?** **3.0010**	Loads **R5**, the constant matrix.
[ALPHA] R [ALPHA] [XEQ] [MEDIT]	**1:1= ?**	Editor for **R0**.
1 [R/S] 1 [R/S] .24 [R/S] .86 [R/S]	**2:1= ?** **3.0010**	Loads **R0**, the coefficient matrix.
[ALPHA] R,R5 [ALPHA]		Specifies the input matrices (coefficient, constant). The solution will go into **R5**.
[XEQ] [MSYS]		Calculates the solution matrix.
[SF] 08		Sets editor for view-only operation.
[ALPHA] R5 [ALPHA] [XEQ] [MEDIT] [R/S] [R/S] [R/S] [R/S] [R/S] [R/S]	 **1:1=186.0000** **1:2=141.0000** **1:3=215.0000** **2:1=88.0000** **2:2=92.0000** **2:3=116.0000** **3.0010**	Displays the results in the solution matrix.

The solution is

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

Alphabetical Function Table

Unless otherwise indicated, each function operates on the matrix (or matrices) named in the Alpha register. When the Alpha register is clear, the function operates on the current matrix.

Matrix Functions

Function Name	Description
AIP (p. 50)	Appends integer part of *x* to Alpha reg.
C<>C (p. 46)	Exchanges columns *k* and *l*.
CMAXAB (p. 48)	Returns maximum absolute value in *k*th column.
CMEDIT (p. 34)	Invokes the complex-matrix editor.
CNRM (p. 48)	Returns the column norm.
CSUM (p. 49)	Finds sums of columns and puts them in a row matrix.
DIM? (p. 33)	Returns the *mmm.nnn* dimension.
FNRM (p. 48)	Returns the Frobenius norm.
I+ (p. 40)	Increments row part of pointer.
I− (p. 40)	Decrements row part of pointer.
J+ (p. 40)	Increments column part of pointer.
J− (p. 40)	Decrements column part of pointer.
M*M (p. 44)	True multiplication (non-scalar) of two matrices.
MAT+ (p. 43)	Adds scalar or matrix to a matrix.
MAT− (p. 43)	Subtracts scalar or matrix from a matrix.
MAT* (p. 44)	Multiplies scalar or matrix by a matrix elementwise.
MAT/ (p. 44)	Divides scalar or matrix into a matrix elementwise.
MATDIM (p. 31)	Dimensions matrix to *mmm.nnn*.
MAX (p. 48)	Returns maximum element.

Matrix Functions (Continued)

Function Name	Description
MAXAB (p. 48)	Returns maximum absolute value of an element.
MDET (p. 45)	Returns determinant.
MEDIT (p. 34)	Invokes the real-matrix editor.
MIN (p. 48)	Returns minimum element.
MINV (p. 45)	Inverts the matrix in place.
MMOVE (p. 46)	Copies source matrix or submatrix to target matrix.
MNAME? (p. 37)	Returns name of current matrix to Alpha reg.
MP (p. 50)	Appends a matrix prompt *rrr*:*ccc*= to Alpha reg.
MR (p. 40)	Recalls current element.
MRC+ (p. 41)	Recalls sequential elements by column.
MRC– (p. 41)	Recalls sequential elements backwards by column.
MRIJ (p. 40)	Recalls pointer *iii.jjj* of current matrix.
MRIJA (p. 40)	Recalls pointer *iii.jjj* .
MRR+ (p. 41)	Recalls sequential elements by row.
MRR– (p. 41)	Recalls sequential elements backwards by row.
MS (p. 40)	Stores current element.
MSC+ (p. 41)	Stores current element by column.
MSIJ (p. 40)	Sets pointer of current matrix to *iii.jjj*.
MSIJA (p. 40)	Sets pointer to *iii.jjj*.
MSR+ (p. 41)	Stores current element by row.
MSWAP (p. 46)	Exchanges two matrices or submatrices.
MSYS (p. 45)	Solves a system of simultaneous equations.
PIV (p. 49)	Returns a column's maximum absolute value that is on or below the diagonal.
R< >R (p. 46)	Exchanges rows *k* and *l*.
R>R? (p. 49)	Tests elementwise whether row *k* is greater than row *l*.
RMAXAB (p. 48)	Returns maximum absolute value in *k*th row.
RNRM (p. 48)	Returns the row norm.
RSUM (p. 49)	Finds sums of rows and puts them in a column matrix.
SUM (p. 49)	Returns sum of all elements.

Matrix Functions (Continued)

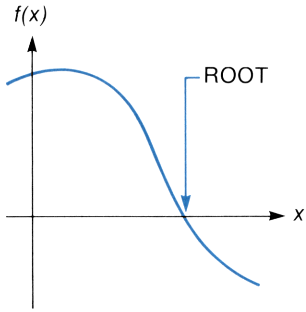
Function Name	Description
SUMAB (p. 49)	Returns sum of absolute values of all elements.
TRNPS (p. 45)	Transposes the matrix in place.
YC+C (p. 49)	Multiplies each element in column *k* by *y*-value and adds product to element in column *l*, replacing the latter.

FINDING THE ROOTS OF AN EQUATION

The SOLVE program finds the roots of an equation of the form

$$f(x) = 0,$$

where x represents a *real root*.*

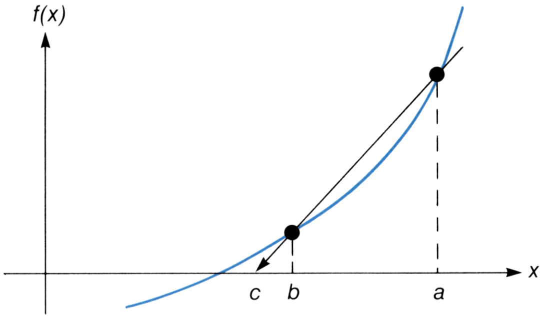


Executing the SOLVE program ($\boxed{\text{SOLVE}}$) employs an advanced numerical technique to find the real roots of a wide range of equations. You supply the equation for the function (in a program) and two initial estimates, and SOLVE does the rest.

Method

SOLVE normally uses the secant method to iteratively find and test x-values as potential roots. It takes the program several seconds to several minutes to do this and produce a result.

* Note that 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 $f(x) - g(x) = 0$.



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

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

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

If this does not yield a root, SOLVE fits a parabola through the function values at a, b, and c, and finds the value d at the parabola's maximum or minimum. The search continues using the secant method, replacing a with d.

If three successive parabolic fits yield no root or $d = b$, the calculator displays **NO**. In the X- and Z-registers remain b and $f(b)$, respectively, with a or c in the Y-register. At this point you could: resume the search where it left off, direct the search elsewhere, decide that $f(b)$ is negligible so that $x = b$ is a root, transform the equation into another equation easier to solve, or conclude that no root exists.

Instructions

In calculating roots, SOLVE repeatedly calls up and executes a program *that you write* for evaluating $f(x)$. You must also provide SOLVE with two initial estimates for x, providing a *range* for it to begin its search for the root.

Realistic estimates greatly facilitate the speedy and accurate determination of a root. If the variable x has a limited range in which it is meaningful and realistic as a solution, it is reasonable to choose initial estimates within this range. (Negative roots, for instance, are often unrealistic for physical problems.)

- SOLVE requires thirteen unused program registers. If enough spare program registers are not available, SOLVE will not run and the error **NO ROOM** results. Execute GTO [·] [·] in Program mode to see how many program registers are available.
- Before running SOLVE you must have a program (stored in program memory or a plug-in module) that evaluates your function $f(x)$ at zero. This program must be named with a *global label*.* SOLVE then iteratively calls your program to calculate successively more accurate estimates of x. Your program can take advantage of the fact that SOLVE fills the stack with its current estimate of x each time it calls your program.
- You then enter two initial estimates for the root, a and b, into the X- and Y-registers.
- Lastly put the name of your program (that evaluates the function) into the Alpha register and then execute [SOLVE].

* This program should *not* include the functions [PASN], [PSIZE], [AK], any card-reader (HP 82104A) functions, or any other function that alters the configuration of the calculator's memory, key assignments, or timer alarms.

When the program stops and the calculator displays a number, the contents of the stack are:

Z = the value of the function at x = root (this value should be zero).*
Y = the previous estimate of the root (should be close to the resulting root).
X = the root (this is what is shown in the display).

If the function that you are analyzing equals zero at more than one value of x, SOLVE stops when it finds any one of these values. To find additional values, key in different initial estimates and execute [SOLVE] again.

Instruction Table for SOLVE

Instructions	Key In:	Display
1. Switch to Program mode and pack memory preparatory to entering a new program. (The display will show you the number of available program registers.)	[PRGM] [GTO] [·] [·]	**00 REG** *nnn*
2. Key in a global, Alpha label as program name for the program describing $f(x)$ for $f(x) = 0$.	[LBL] *global label*	**01 LBL**^{T}*label*
3. Key in the lines of the program and end the program with a [RTN] instruction.	⋮ [RTN]	
4. Check that program memory is large enough to run SOLVE ($nnn \geqslant 13$).* Then switch out of Program mode.	[GTO] [·] [·] [PRGM]	**00 REG** ***nnn***
5. Put the name of your program from step 2 into the Alpha register.	[ALPHA] *global label* [ALPHA]	
6. Enter the range for the initial search for x:	*a* [ENTER↑] *b*	*a* *b*

* If the contents of the Z-register are *not* zero, then the X-register does not contain the exact root. Instead, the contents of X and Y are close estimates of the root, bracketing a change in the sign of the function's value.

Instruction Table for SOLVE (Continued)

Instructions	Key In:	Display
7. Execute [SOLVE]. The program runs up to several minutes and then returns the resulting root. If no root is found, the display is **NO**. 8. To search for another root, repeat steps 6 and 7.	[SOLVE]†	x

* If *nnn* is not ⩾13, then use [SIZE] to allocate more memory to program registers, or else delete programs. Refer to the HP-41 owner's manual for instructions.

† To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

Remarks

Pressing [R/S] aborts the SOLVE program.

Examples

Find the roots of the equation $f(x) = x^2 - 3x - 10 = 0$.

First write a program called TEST to define the function. Then, before executing [SOLVE], put the name of this program into the Alpha register and enter your initial estimates for the root.

Using Horner's method you can rewrite $f(x)$ so that it is more efficiently programmable: $f(x) = (x - 3)x - 10$. (Note that you could also find this root algebraically.) Since the SOLVE program fills the stack with the current estimate of x before calling TEST, TEST can obtain x from the stack when TEST runs.

Keystrokes	Display	
[FIX] 4		Sets the display format used here.
[PRGM] [GTO] [·] [·]	00 REG *nnn*	Program mode; ready to enter a program to evaluate $(x - 3)x - 10$.
[LBL] [ALPHA] TEST [ALPHA]	01 LBL^{T}TEST	Global Alpha label "TEST".
3	02 3_	

Keystrokes	Display	
[−]	03 −	$(x - 3)$
[×]	04 *	$(x - 3)x$
10	05 10_	
[−]	06 −	$(x - 3)x - 10$
[RTN]	07 RTN	End of program defining $f(x)$.
[GTO] [·] [·]	00 REG *nnn*	Number of available program registers (should be $\geqslant 13$).
[PRGM]		Exits Program mode.
[ALPHA] TEST [ALPHA]	TEST_	Puts "TEST" (your program's name) into the Alpha register. This is the necessary first step to running SOLVE.
0 [ENTER↑] 10	10_	Enters initial estimates of zero and ten. Now you're ready to execute [SOLVE].
[XEQ] [SOLVE]	5.0000	Runs the SOLVE program; finds a root of $x = 5.0000$ (in about 12 seconds).

Check that 5.0000 is indeed a root of $f(x) = 0$ by checking the Z-register. Then check for a second root (which is common in quadratic equations) by specifying new initial estimates of 0 and −10 to look for a negative root.

Keystrokes	Display	
[R↓] [R↓]	0.0000	Displays first the Y-register, then the Z-register. Since $f(5) = 0$, 5 is a good root.
0 [ENTER↑] 10 [CHS]	−10_	New initial estimates to look for a second root.
[XEQ] [SOLVE]	−2.0000	Second root.
[R↓] [R↓]	0.0000	This root is also good.

Here is a problem whose root cannot be found algebraically. If champion ridget hurler Chuck Fahr throws a ridget with an upward velocity of 50 meters/second, then how long does it take for it to reach the ground again? Solve for t in the equation

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

Assume h in meters and t in seconds. Naturally we are only interested in a *positive* root, t.

As in the previous example, the program you write to define the function can take advantage of the fact that the stack is filled with the current estimate of x before calling your program.

Keystrokes	**Display**	
PRGM GTO · ·	00 REG *nnn*	
LBL ALPHA HIGH ALPHA	01 LBL^THIGH	Names this program "HIGH" with a global label.
20 CHS	02 −20_	
÷	03 /	$-t/20$
e^x	04 E↑X	
CHS	05 CHS	$-e^{-t/20}$
1	06 1_	
+	07 +	$1-e^{-t/20}$
5000	08 5000_	
×	09 *	$5000\,(1-e^{-t/20})$
x≶y	10 X<>Y	
200	11 200_	
×	12 *	$200t$
−	13 −	
RTN	14 RTN	We now have the full equation so the program is done: $5000(1 - e^{-t/20}) - 200t$
GTO · ·	00 REG *nnn*	Is $nnn \geqslant 13$?
PRGM		Exits Program mode.
ALPHA HIGH ALPHA		Puts your program's name into the Alpha register.
5 ENTER↑ 6	6_	Example of initial estimates for t.
XEQ SOLVE	9.2843	The root $t = 9.2843$ seconds.
R↓ R↓	0.0000	Shows that $h(9.2843) = 0$.

When No Root Is Found

It is possible that an equation has no real roots. In this case, the calculator displays **NO** instead of a numeric result. This would happen, for example, if you tried to solve the equation

$$|x| = -1,$$

which has no solution since the absolute value function is never negative.

There are three general types of errors that stop SOLVE from running:

- If repeated iterations seeking a root produce a constant nonzero value for the specified function, the calculator displays **NO**.
- If numerous samples indicate that the *magnitude* of the function appears to have a nonzero minimum value in the area being searched, the calculator displays **NO**.
- If an improper argument is used in a mathematical operation as part of your program, the calculator displays **DATA ERROR**.

Programming Information

You can incorporate SOLVE as part of a larger program you create. Be sure that your program provides initial estimates in the X- and Y-registers just before it executes [SOLVE]. Remember also that SOLVE will look in the Alpha register for the name of the program that calculates your function.

If the execution of SOLVE in your program produces a root, then your program will proceed to its next line. If *no* root results, the next program line will be skipped. (This is the "do if true" rule of HP-41 programming.) Knowing this, you can write your program to handle the case of SOLVE not finding a root, such as by choosing new initial estimates or changing a function parameter.

SOLVE uses one of the six pending subroutine returns that the calculator has, leaving five returns for a program that calls SOLVE.

Note that SOLVE cannot be used recursively (calling itself). If it does, the program stops and displays **RECURSION**. You *can* use SOLVE with INTEG, the integration program.

References

"Using SOLVE Effectively," *HP-15C Advanced Functions Handbook*, Hewlett-Packard Co., 1982.

Kahan, W.M., "Personal Calculator Has Key to Solve Any Equation $f(x)=0$," *Hewlett-Packard Journal*, 30:12, December 1979.

POLYNOMIAL SOLUTIONS AND EVALUATIONS

The PLY program can be used to find the roots of a polynomial with real coefficients of degree 2 through 5, or to evaluate an equation of degree 2 through 20.

The polynomial equation can be represented as:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0,$$

where $n = 2, 3, 4,$ or 5.

Polynomials can also be evaluated for arbitrary values of x. This is useful for plotting polynomials and using data correlations based on polynomials.

When the program is started, the user must specify the degree (n) of the polynomial. The calculator then prompts the user for the coefficients a_n, ..., a_1, a_0. Zero must be input for those coefficients that are equal to 0. Registers 00 through 05 are used to store the coefficients. (Registers 00 through 20 are used for coefficients when *evaluating* a polynomial of degree up to 20.)

Equations

In finding the roots the first step of the routine is to divide all coeffiecients by a_n to produce an equation of the form $x^n + a'_{n-1} x^{n-1} + \ldots + a_0' = 0$. The divisor is retained in register a_n for use in evaluating the polynomial for arbitrary values of x.

The routines for third and fifth degree equations use an iterative process to find one real root of the equation. This routine requires that the constant term a_0 not be zero for these equations. (If $a_0 = 0$, then zero is a real root. The equation can be reduced by one order by factoring out x.) After one root is found, synthetic division is performed to reduce the original equation to a second or fourth degree equation.

To solve a fourth degree equation, it is first necessary to solve the cubic equation

$$y^3 + b_2 y^2 + b_1 y + b_0 = 0,$$

where
$$\begin{aligned} b_2 &= -a_2 \\ b_1 &= a_3 a_1 - 4a_0 \\ b_0 &= a_0 (4a_2 - a_3^2) - a_1^2. \end{aligned}$$

Let y_0 be the largest real root of the above cubic.

Then, the fourth degree equation is reduced to two quadratic equations:

$$x^2 + (A + C)x + (B + D) = 0$$

$$x^2 + (A - C)x + (B - D) = 0$$

where $A = \dfrac{a_3}{2}$, $B = \dfrac{y_0}{2}$, $D = \sqrt{B^2 - a_0}$, $C = \sqrt{A^2 - a_2 + y_0}$

Roots of the fourth degree equation are found by solving the two quadratic equations.

A quadratic equation $x^2 + a_1x + a_0 = 0$ is solved by the formula

$$x_{1,2} = -\frac{a_1}{2} \pm \sqrt{\frac{a_1^2}{4} - a_0}.$$

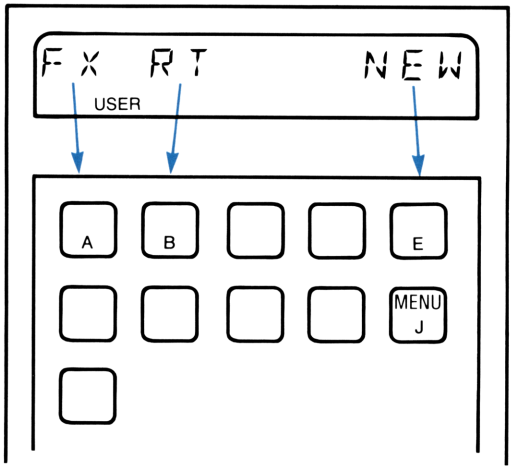
If $D = a_1^2/4 - a_0 > 0$, the roots are real; if $D < 0$, the roots are complex, being $u \pm iv = -(a_1/2) \pm i\sqrt{-D}$.

A real root is displayed as a single number. Complex roots always occur in pairs of the form $u \pm iv$, and are labeled in the output.

Long execution times can be expected for equations of degree 3, 4, or 5, as these use an iterative routine once or more.

Instructions

Once you have entered your variables, this menu shows you which key corresponds to which function in PLY. Press [J] to recall this menu to the display at any time. This will not disturb the program in any way.



To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then continue the program by pressing [R/S]. (However, you do not *need* to clear the program's display before performing calculations.)

Instruction Table for PLY

Instructions	Key In:	Display
		Size: 023
1. Start the PLY program.	[XEQ] [PLY] *	**DEGREE=?**
2. Key in the degree of the polynomial (n = 2,3,4,5 for root finding; up to 20 if evaluating only).	n [R/S]	**a**n**=?**
3. Input coefficient a_n of the polynomial. (Coefficients = 0 must also be entered.) Repeat until display asks for a_0.	a_n [R/S] ⋮ a_1	**a**(n − 1)**=?** ⋮ **a0=?**
4. Input coefficient a_0.	a_0 [R/S]	**FX RT NEW**
5. To evaluate the polynomial for x, use **FX**. You can repeat this step for new values of x.	x [A] (**FX**) [R/S] †	**F<X>=**$f(x)$ **FX RT NEW**

Instruction Table for PLY (Continued)

Instructions	Key In:	Display
6. To find the roots of the polynomial, use **RT** and then [R/S] to display successive roots.	[B] (**RT**) [R/S] ⋮ [R/S] [R/S] [R/S] [R/S] [R/S] †	**ROOT**=*root 1* **ROOT**=*root 2* ⋮ **U**=*u-value* **V**=*v-value* **U**=*u-value* **V**=−*v-value* **FX RT NEW**
7. To work out a new polynomial, choose **NEW** ([E]) and return to step 2.	[E] (**NEW**)	**DEGREE=?**

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

† This keystroke is unnecessary if you have a printer attached because the printer automatically prints the results and then displays the menu.

Note: This program can calculate incorrect roots due to rounding off of intermediate results. Incorrect roots normally occur only for *real* roots. To check the calculated root, rerun PLY to evaluate a polynomial (step 5). Input the root x that you want to check. If the result is a very small number close to zero, then the root is correct.

Remarks

If you set flag 06 ([SF] 06) just before step 6, then the roots found in step 6 will be stored as they are found, starting in R_{24} and in the order real, imaginary. (Real roots store a zero imaginary part.)

This program uses local Alpha labels (as explained in the owner's manual for the HP-41) assigned to keys [A], [B], [E], and [J]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

Examples

Find the roots of $x^5 - x^4 - 101x^3 + 101x^2 + 100x - 100 = 0$.

Keystrokes	Display	
FIX 4		Sets the display format used here.
XEQ SIZE 023		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 023.
XEQ PLY	DEGREE=?	
5 R/S	a5=?	
1 R/S	a4=?	
1 CHS R/S	a3=?	
101 CHS R/S	a2=?	
101 R/S	a1=?	
100 R/S	a0=?	
100 CHS R/S	FX RT NEW	
B (**RT**)	ROOT=10.0000	Root 1.
R/S	ROOT=1.0000	Root 2.
R/S	ROOT=1.0000	Root 3.
R/S	ROOT=−1.0000	Root 4.
R/S	ROOT=−10.0000	Root 5.

Solve $4x^4 - 8x^3 - 13x^2 - 10x + 22 = 0$.

Keystrokes	Display	
J	FX RT NEW	Displays the menu (optional step).
E (**NEW**)	DEGREE=?	Prompts for a new polynomial (after the one in Example 1.)
4 R/S	a4=?	
4 R/S	a3=?	
8 CHS R/S	a2=?	
13 CHS R/S	a1=?	
10 CHS R/S	a0=?	
22 R/S	FX RT NEW	Displays the menu.
B (**RT**)	U=−1.0000	
R/S	V=1.0000	Roots 1 and 2 are $-1.00 \pm 1.00i$.
R/S	U=−1.0000	
R/S	V=−1.0000	
R/S	ROOT=3.1180	Root 3.
R/S	ROOT=0.8820	Root 4.

Evaluate the following polynomial at $x = 2.5$ and $x = -5$.

$$f(x) = x^5 + 5x^4 - 3x^2 - 7x + 11$$

Keystrokes	Display	
[J]	FX RT NEW	Displays the menu (optional step).
[E] (NEW)	DEGREE=?	Prompts for a new polynomial.
5 [R/S]	a5=?	
1 [R/S]	a4=?	
5 [R/S]	a3=?	
0 [R/S]	a2=?	
3 [CHS] [R/S]	a1=?	
7 [CHS] [R/S]	a0=?	
11 [R/S]	FX RT NEW	
2.5 [A] (FX)	F<X>=267.7188	
5 [CHS] [R/S]	F<X>=−29.0000	

Programming Information

The subroutine RTS can be used in your own programs. It finds the real and complex roots of a polynomial of degree 2 to 5.

Minimum Size to Run RTS: SIZE 023, unless flag 6 is set. If the roots are to be stored, then the number of data-storage registers needed is 24 + 2(degree).

Flags Used: 00, 03, 05, 06, 21

Subroutine: RTS

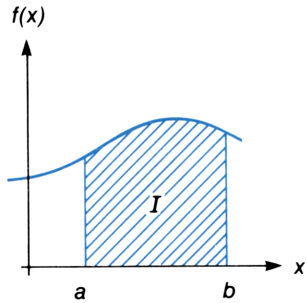
Initial Registers	Final Registers	Flags to Initialize
$R_{00} = a_0$	$R_{00} = a_0/a_5$	SF 00
$R_{01} = a_1$	$R_{01} = a_1/a_5$	CF 03
$R_{02} = a_2$	$R_{02} = a_2/a_5$	CF 05
$R_{03} = a_3$	$R_{03} = a_3/a_5$	SF 06 to save roots
$R_{04} = a_4$	$R_{04} = a_4/a_5$	CF 06 to not save roots
$R_{05} = a_5$	$R_{05} = a_5$	SF 21 to stop when displaying results
	$R_{06} \ldots R_{21}$ = scratch	CF 21 to not stop when displaying results
R_{22} = degree of equation	R_{22} = degree of equation	
R_{23} = pointer		
If flag 06 is set:		
	R_{24}, R_{25} = root 1	
	R_{26}, R_{27} = root 2	
	R_{28}, R_{29} = root 3	
	R_{30}, R_{31} = root 4	
	R_{32}, R_{33} = root 5	

Comments. To use RTS, load the coefficients in R_{00}–R_{05}, the degree in R_{22}, set flag 06 to store the roots, clear flags 03 and 05, and set flag 00. If roots are stored they are stored with real and imaginary parts; a real root has a zero imaginary part.

NUMERICAL INTEGRATION

The INTEG program finds the definite integral, I, of a function $f(x)$ within the interval bounded by a and b. This is expressed mathematically and graphically as

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



Executing the INTEG program (INTEG) employs an advanced numerical technique to find the definite integral of a function. You supply the equation for the function (in a program) and the interval of integration, and INTEG does the rest.

Method

The algorithm for INTEG uses a Romberg method for accumulating the value of an integral. The algorithm evaluates $f(x)$ at many values of x between the limits of integration. It takes the program from several seconds to several minutes to do this and produce a result.

Several refinements make the algorithm more effective. For instance, instead of using uniformly spaced samples, which can induce a kind of resonance producing misleading results when the integrand is periodic, INTEG uses samples that are spaced nonuniformly. Another refinement is that INTEG uses extended precision (13 significant digits) to accumulate the internal sums. This allows thousands of samples to be accurately accumulated, if necessary.

A calculator using numerical integration can almost never calculate an integral precisely. However, there is a convenient way for you to specify how much error is tolerable. You can set the display format according to how many figures are accurate in the integrand $f(x)$. A setting of [FIX] 2 tells the calculator that decimal digits beyond the second one can't matter, so the calculator need not waste time estimating the integral with unwarranted precision. Refer to the heading, "Accuracy of INTEG."

Instructions

In calculating integrals, INTEG repeatedly executes a program *that you write* for evaluating $f(x)$. You must also provide INTEG with two limits for x, providing an interval of integration.

- INTEG requires 32 unused program registers. If enough spare program registers are not available, INTEG will not run and the error **NO ROOM** results. Execute [GTO] [·] [·] in Program mode to see how many program registers are available.
- Before running INTEG you must have a program (stored in program memory or a plug-in module) that evaluates your function $f(x)$. This program must be named with a *global label*.* Your program can take advantage of the fact that INTEG fills the stack with its current estimate of x each time it calls your program.
- You then enter the two limits, a and b, into the X- and Y-registers.
- Lastly put the name of your program (that evaluates the function) into the Alpha register and then execute [INTEG].

When the program stops and the calculator displays the integral, the contents of the stack are:

T = the lower limit of the integration, a.
Z = the upper limit of the integration, b.
Y = the uncertainty of the approximation of the integral.
X = the approximation of the integral (this is what is shown in the display).

---

* This program should *not* include the functions [PASN], [PSIZE], [AK], any card-reader (HP 82104A) functions, or any other function that alters the configuration of the calculator's memory, key assignments, or timer alarms.

Instruction Table for INTEG

Instructions	Key In:	Display
1. Switch to Program mode and pack memory preparatory to entering a new program.	[PRGM] [GTO] [.] [.]	00 REG *nnn*
2. Key in a global, Alpha label as program name for the program describing *f*(*x*).	[LBL] *global label*	01 LBL^T*label*
3. Key in the lines of the program and end the program with a [RTN] instruction.	⋮ [RTN]	
4. Check that program memory is large enough to run INTEG (*nnn*⩾32).* Then switch out of Program mode.	[GTO] [.] [.] [PRGM]	00 REG *nnn*
5. Put the name of your program from step 2 into the Alpha register.	[ALPHA] *global label* [ALPHA]	
6. Enter the limits for the initial search for the integral:	*a* [ENTER↑] *b*	*a* *b*
7. Set the display format to determine the accuracy of the result.	[FIX] *n* or [SCI] *n* or [ENG] *n*	
8. Execute [INTEG]. The program runs up to several minutes and then returns the resulting integral.	[INTEG]†	*integral*
9. To repeat this calculation using a different level of accuracy, set a new display format, roll down the stack to retrieve the original upper and lower limits, and re-execute [INTEG].	[FIX] *n* or [SCI] *n* or [ENG] *n* [R↓] [R↓] [INTEG]	*b* *integral*

* If *nnn* is not ⩾32, then use [SIZE] to allocate more memory to program registers, or else delete programs. Refer to the HP-41 owner's manual for instructions.

† To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

Remarks

Pressing [R/S] aborts the INTEG program.

Example 1

The Bessel function of the first kind of order 0 can be expressed as

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

Find

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

First write a program to define the integrand. Make sure the calculator is set to Radians mode to calculate these trigonometric functions. Then, before executing [INTEG], put the name of your program into the Alpha register and enter the limits of integration. Once you've found the integral, don't forget to multiply it by $1/\pi$.

Keystrokes	Display	
[FIX] 4		Sets the display format used here.
[PRGM] [GTO] [·] [·]	**00 REG** *nnn*	Program mode; ready to enter a program to evaluate cos(sin θ).
[LBL] [ALPHA] J01 [ALPHA]	**01 LBL^{T}J01**	Global Alpha label "J01".
[SIN]	**02 SIN**	sin θ.
[COS]	**03 COS**	cos(sin θ).
[RTN]	**04 RTN**	End of program defining $f(x)$.
[GTO] [·] [·]	**00 REG** *nnn*	Number of available program registers; is $nnn \geqslant 32$?
[PRGM]		Exits Program mode.
[ALPHA] J01 [ALPHA]	**J01**	Puts "J01" (your program's name) into the Alpha register. This is the necessary first step to running INTEG.
0 [ENTER↑] [π]	**3.1416**	Enters integration limits of zero and π.
[RAD]	**3.1416**	Sets Radians mode. Now you're ready to execute [INTEG].

Keystrokes	Display	
XEQ INTEG	2.4040	Runs INTEG and returns the integral (in about 25 seconds). To complete the equation, don't forget to multiply by the constant outside the integral.
π	3.1416	
÷	0.7652	$J_0(1)$.

Accuracy of INTEG

Since the calculator cannot compute the value of an integral exactly, it *approximates* it. The accuracy of this approximation depends on the accuracy of the integrand's function itself as calculated by your program.* This is affected by round-off error in the calculator and the accuracy of empirical constants.

To specify the accuracy of the function, set the display format (FIX n, SCI n, or ENG n) so that n is no greater than the number of decimal digits that you consider accurate in the function's values. If you set n smaller, the calculator will compute the integral more quickly, but it will also presume that the function is accurate to no more than the number of digits shown in the display format.†

At the same time that the INTEG program returns the resulting integral to the X-register (the display), it returns the *uncertainty* of that approximation to the Y-register.‡ To view this uncertainty value, press x≷y.

If the uncertainty of an approximation is greater than what you choose to tolerate, you can decrease it by specifying more digits in the display format and rerunning INTEG.

---

* While integrals of functions with certain characteristics such as spikes or rapid oscillations might be calculated inaccurately, these functions are rare.

† SCI and ENG determine an uncertainty in the function that is *proportional* to the function's magnitude, while FIX determines an uncertainty that is *independent* of the function's magnitude.

‡ No algorithm for numerical integration can compute the exact difference between its approximation and the actual integral. But this algorithm estimates an *upper bound* on this difference, which is returned as the *uncertainty* of the approximation.

To rerun INTEG for the same problem but with a different display format, you do not need to re-enter the limits of integration (*if* you have not made any calculations subsequent to finding the integral). Since they end up in the T- and Z-registers (as shown under "Instructions"), just press [R↓] [R↓] to retrieve them, then execute [INTEG] again.

Example 2

With the display format set to [SCI] 2, calculate the integral in the expression for $J_0(1)$ in example 1. Check the uncertainty of this result. Then calculate a result accurate to four decimal places instead of only two, and check its uncertainty. (Make sure that Radians mode is still set by checking for the **RAD** annunciator, which should be on.) You will have to re-enter the limits of integration for the *first* computation only.

Keystrokes	**Display**	
[SCI] 2		Sets scientific notation; two decimal places of accuracy.
0 [ENTER↑] [π]	3.14 00	Enters the lower (0) and upper limits (π).
[XEQ] [INTEG]	2.40 00	The integral, accurate to two decimal places.
[x⪤y]	1.57 −03	The uncertainty of the integral.
[SCI] 4	1.5708 −03	Sets four decimal places of accuracy.
[R↓] [R↓]	3.1416 00	Roll down stack until upper limit appears.
[XEQ] [INTEG]	2.4039 00	Integral accurate to four decimal places.
[x⪤y]	1.5708 −05	Uncertainty (much smaller).

Programming Information

You can incorporate INTEG as part of a larger program you create. Be sure that your program provides upper and lower limits in the X- and Y-registers just before it executes [INTEG]. Remember also that INTEG will look in the Alpha register for the name of the program that calculates your function.

INTEG uses one of the six pending subroutine returns that the calculator has, leaving five returns for a program that calls INTEG.

Note that INTEG cannot be used recursively (calling itself). If it is, the program stops and displays **RECURSION**. You *can* use INTEG with SOLVE. A routine that combines INTEG and SOLVE requires 32 available program registers to operate.

References

"Working with $\boxed{\int_y^x}$," *HP-15C Advanced Functions Handbook,* Hewlett-Packard Co., 1982.

Kahan, W.M., "Handheld Calculator Evaluates Integrals," *Hewlett-Packard Journal*, 31:8, August 1980.

DIFFERENTIAL EQUATIONS

The DIFEQ program solves first- and second-order differential equations by the fourth-order Runge-Kutta method. A first-order equation is of the form $y' = f(x, y)$, with initial values x_0, y_0; a second-order equation is of the form $y'' = f(x, y, y')$, with initial values x_0, y_0, y_0'.

In either case, the function $f(x)$ may be keyed into program memory using any *global* label (maximum of six characters), and should assume that x and y are in the X- and Y-registers respectively; y' will be in the Z-register for second-order equations. The DIFEQ program uses registers 00 through 07. The remaining registers are available for defining the function.

The solution is a numerical solution which calculates y_i for $x_i = x_0 + ih$ $(i = 1, 2, 3, \ldots)$, where h is an increment specified by the user. The value for h can be changed at any time during program execution by storing $h/2$ in Register 01. This allows solution of the equation arbitrarily close to a pole ($y \rightarrow \pm\infty$).

Equations

First order:

$$y_{i+1} = y_i + \frac{1}{6}(c_1 + 2c_2 + 2c_3 + c_4)$$

where

$$c_1 = hf(x_i, y_i)$$

$$c_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{c_1}{2}\right)$$

$$c_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{c_2}{2}\right)$$

$$c_4 = hf(x_i + h, y_i + c_3)$$

Second order:

$$y_{i+1} = y_i + h\left[y_i' + \frac{1}{6}(k_1 + k_2 + k_3)\right]$$

$$y'_{i+1} = y_i' + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = hf(x_i, y_i, y_i')$$

$$k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}y_i' + \frac{h}{8}k_1, y_i' + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}y_i' + \frac{h}{8}k_2, y_i' + \frac{k_2}{2}\right)$$

$$k_4 = hf\left(x_i + h, y_i + hy_i' + \frac{h}{2}k_3, y_i' + k_3\right)$$

Instructions

When you are inputting values for a second-order solution, the values for x_0 and y_0 must be input before the value of y_0'. All values must be input, including values of zero.

Note that a value for h, the step size, that is too large can generate incorrect results.*

* You can check a result by working backward from the result to the initial condition using $-h$. If you don't get the correct initial value, then rerun DIFEQ with a smaller h.

Instruction Table for DIFEQ

Size: 008

Instructions	Key In:	Display
1. Prepare to load function $f(x, y, y')$.	[GTO] [·] [·]	
2. Switch to Program mode.	[PRGM]	
3. Load function under desired global, Alpha label. Add [RTN]	[LBL] *function label* ⋮ [RTN]	
4. Exit Program mode.	[PRGM]	
5. Start the program.	[XEQ] [DIFEQ]*	**NAME?**
6. Key in *function label* (from step 3).	*function label* [R/S]	 **ORDER=?**
7. Key in order of the differential equation (1 or 2).	*order* [R/S]	**STEP SIZE=?**
8. Key in step size (h).	h [R/S]	**X0=?**
9. Input initial value for x.	x_0 [R/S]	**Y0=?**
10. Input initial value for y.	y_0 [R/S]	x_1 (first-order equation) or **Y0.=?** (second-order equation)
11. For a second-order equation, key in initial value of y'.	y_0' [R/S]	x_1
12. Output successive values of x and y.	[R/S] [R/S] [R/S]	y_1 x_2 y_2 etc.

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

Examples

Using the function label FX, solve numerically the first-order differential equation

$$y' = \frac{\sin x + \tan^{-1}(y/x)}{y - \ln(\sqrt{x^2 + y^2})}$$

where $x_0 = y_0 = 1$. Let $h = 0.5$. The angular mode must be set to Radians, and three additional storage registers are necessary to define the function.

Keystrokes	Display	
FIX 4		Sets the display format used here.
XEQ SIZE 011		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE $\geqslant$ 011.
PRGM		
GTO . .		
LBL ALPHA FX ALPHA		
XEQ RAD		
STO 08		
x≶y		
STO 09		
x≶y		
R→P		
LN		
STO 10		
R↓		
RCL 08		
SIN		
+		
RCL 09		
RCL 10		
−		
÷		
XEQ DEG		
RTN		
PRGM		

Keystrokes	Display	
[XEQ] [DIFEQ]	NAME?	
FX [R/S]	ORDER=?	
1 [R/S]	STEP SIZE=?	
.5 [R/S]	X0=?	
1 [R/S]	Y0=?	
1 [R/S]	1.5000	x_1
[R/S]	2.0553	y_1
[R/S]	2.0000	x_2
[R/S]	2.7780	y_2
[R/S]	2.5000	x_3
[R/S]	3.2781	y_3
	etc.	

Using the function label DIF, solve the second-order equation

$$(1 - x^2)y'' + xy' = x,$$

where $x_0 = y_0 = y_0' = 0$ and $h = 0.1$.

Rewrite the equation as

$$y'' = \frac{x(1 - y')}{1 - x^2} = \frac{x(y' - 1)}{x^2 - 1} \qquad x \neq 1$$

Keystrokes	Display
[PRGM]	
[GTO] [·] [·]	
[LBL] [ALPHA] DIF [ALPHA]	
[STO] 08	
[R↓] [R↓]	
1 [−]	
[RCL] 08	
[×]	
[LASTx]	
[x^2]	
1 [−] [÷]	
[RTN]	
[PRGM]	

Keystrokes	Display	
[XEQ] [DIFEQ]	NAME?	
DIF [R/S]	ORDER=?	
2 [R/S]	STEP SIZE=?	
.1 [R/S]	X0=?	
0 [R/S]	Y0=?	
0 [R/S]	Y0.=?	
0 [R/S]	0.1000	x_1
[R/S]	0.0002	y_1
[R/S]	0.2000	x_2
[R/S]	0.0013	y_2
[R/S]	0.3000	x_3
[R/S]	0.0046	y_3
[R/S]	0.4000	x_4
[R/S]	0.0109	y_4
	etc.	

OPERATIONS WITH COMPLEX NUMBERS

This collection of operations provides the ability to do chained calculations involving complex numbers in rectangular form. The four operations of complex arithmetic ($+$, $-$, $\times$, $\div$) are provided, as well as several of the most used functions of complex variables z and w ($|z|$, $1/z$, z^n, $z^{1/n}$, e^z, $\ln z$, $\sin z$, $\cos z$, $\tan z$, a^z, $\log_a z$, $z^{1/w}$, and z^w). Functions and operations can be mixed in the course of a calculation to allow evaluation of expressions such as $z_3/(z_1 + z_2)$, $e^{z_1 z_2}$, $|z_1 + z_2| + |z_2 - z_3|$, etc., where z_1, z_2, and z_3 are complex numbers of the form $x + iy$.

For repeated use of these operations, the user can reassign the individual programs to selected keys of the calculator and create an appropriate overlay. One reasonable set of reassignments might include:

[ASN] [SINZ] [SIN]
[ASN] [LNZ] [LN]
[ASN] [C+] [+]
[ASN] [C−] [−]
[ASN] [CINV] [1/x]

The logic system for these functions is a variation on the regular memory stack for the HP-41. Instead of holding four real numbers, this stack holds two complex numbers. Let the bottom register of the complex stack be ξ and the top register τ. These are analogous to the X- and T-registers in the calculator's own four-register stack.* A complex number z is input to the ξ-register by the keystrokes z_y [ENTER↑] z_x. Upon input of a second complex number w ([ENTER↑] w_y [ENTER↑] w_x), z is lifted into τ and w is placed in ξ. The previous contents of τ are lost.

---

* Each register of the complex stack must actually hold two real numbers—the real part and the imaginary part of its complex contents. Thus, it takes two of the calculator registers to represent one register in the complex stack. In this discussion, we will treat the two registers containing a complex number as though they were one register.

Memory Stacks

T	t
Z	z
Y	y
X	x

Regular stack

τ	iz_y	z
	z_x	
ξ	iw_y	w
	w_x	

Complex stack

Functions operate on the ξ-register, and the result (except for $|z|$, which returns a real number) is left in ξ. Arithmetic operations involve both the ξ- and τ-registers; the result of the operation is left in ξ.

These functions use registers 00 through 04.

Equations

Let

$$z_k = x_k + iy_k = r_k e^{i\theta_k};\ k = 1, 2$$
$$z = x + iy = re^{i\theta}$$

Let the result in each case be $u + iv$.

$$z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2)$$
$$z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2)$$
$$z_1 z_2 = r_1 r_2 e^{i(\theta_1 + \theta_2)}$$
$$z_1/z_2 = \frac{r_1}{r_2} e^{i(\theta_1 - \theta_2)}$$
$$|z| = \sqrt{x^2 + y^2}$$
$$1/z = \frac{x}{r^2} - i\frac{y}{r^2}$$
$$z^n = r^n e^{in\theta}$$
$$z^{1/n} = r^{1/n} e^{i\left(\frac{\theta}{n} + \frac{360k}{n}\right)},\ k = 0, 1, \ldots, n-1$$

(All n roots will be output, $k = 0, 1, \ldots, n-1$.)

$$e^z = e^x (\cos y + i \sin y), \text{ where } y \text{ is in radians}$$
$$\ln z = \ln r + i\theta, \text{ where } z \neq 0$$
$$a^z = e^{z \ln a}, \text{ where } a > 0 \text{ and real}$$

$$\log_a z = \frac{\ln z}{\ln a}\ ,\ \text{where } a > 0 \text{ and real, } z \neq 0$$

$$z^w = e^{w \ln z},\ \text{where } z \neq 0,\ w \text{ is complex}$$

$$z^{1/w} = e^{\ln z / w},\ \text{where } z \neq 0,\ w \text{ is complex and } w \neq 0$$

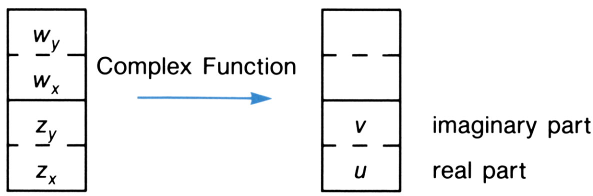
$$\sin z = \sin x \cosh y + i \cos x \sinh y,\ \text{angles in radians}$$

$$\cos z = \cos x \cosh y - i \sin x \sinh y,\ \text{angles in radians}$$

$$\tan z = \frac{\sin 2x + i \sinh 2y}{\cos 2x + \cosh 2y}\ ,\ \text{angles in radians}$$

Instructions

Typical Input and Output



Input
(z, or w and z)

Result
($u + iv$)

Instruction Table for Complex Arithmetic Functions

Instructions	Key In:	Display
		Size: 005
Complex Arithmetic Functions		
1. Key in the first complex number ($z_x + iz_y$).	z_y ENTER↑ z_x ENTER↑	z_y z_x
2. Key in the second complex number ($w_x + iw_y$).	w_y ENTER↑ w_x	w_y w_x
3. Select one of four operations:		

Instruction Table for Complex Arithmetic Functions (Continued)

Instructions	Key In:	Display
■ Addition	XEQ C+ R/S	U=*u-value* V=*v-value*
■ Subtraction	XEQ C− R/S	U=*u-value* V=*v-value*
■ Multiplication	XEQ C× R/S	U=*u-value* V=*v-value*
■ Division	XEQ C÷ R/S	U=*u-value* V=*v-value*
4. The result of the operation remains in the stack; return to step 2 for further arithmetic.		
Complex Functions with One Complex Number		
1. Key in the complex number ($z_x + iz_y$).	z_y ENTER↑ z_x	z_y z_x
2. Select one of these operations:		
■ SINZ (sin z)	XEQ SINZ R/S	U=*u-value* V=*v-value*
■ COSZ (cos z)	XEQ COSZ R/S	U=*u-value* V=*v-value*
■ TANZ (tan z)	XEQ TANZ R/S	U=*u-value* V=*v-value*
■ MAGZ (magnitude, $\|z\|$)	XEQ MAGZ	R=*magnitude*
■ CINV ($1/z$)	XEQ CINV R/S	U=*u-value* V=*v-value*
■ e↑Z (e^z)	XEQ e↑Z R/S	U=*u-value* V=*v-value*
■ LNZ (ln z)	XEQ LNZ R/S	U=*u-value* V=*v-value*
■ Z↑N (z^n, where n is an integer)	ENTER↑ n XEQ Z↑N R/S	n U=*u-value* V=*v-value*
■ Z↑1/N ($z^{1/n}$) Note that n roots ($u + iv$) will be found.	ENTER↑ n XEQ Z↑1/N R/S	n U=*u-value* V=*v-value*

Instruction Table for Complex Arithmetic Functions (Continued)

Instructions	Key In:	Display
■ [a↑Z] (a^z, where a is real)	[ENTER↑] a [XEQ] [a↑Z] [R/S]	a U=*u-value* V=*v-value*
■ [LOGZ] ($\log_a z$, where a is real)	[ENTER↑] a [XEQ] [LOGZ] [R/S]	a U=*u-value* V=*v-value*
Complex Functions with Two Complex Numbers		
1. Key in the first complex number ($z_x + iz_y$).	z_y [ENTER↑] z_x [ENTER↑]	z_y z_x
2. Key in the second complex number ($w_x + iw_y$).	w_y [ENTER↑] w_x	w_y w_x
3. Select one of these operations:		
■ [Z↑W] (z^w)	[XEQ] [Z↑W] [R/S]	U=*u-value* V=*v-value*
■ [Z↑1/W] ($z^{1/w}$)	[XEQ] [Z↑1/W] [R/S]	U=*u-value* V=*v-value*

Remarks

When flag 04 is set, the individual complex operations (which are actually programs) can be accessed as subroutines in your own programs. Complex results are returned to the X- (real part) and Y- (imaginary part) registers.

Examples

Evaluate the expression

$$\frac{z_1}{z_2 + z_3},$$

where $z_1 = 23 + 13i$, $z_2 = -2 + i$, $z_3 = 4 - 3i$.

Suggestion: since the program can remember only two numbers at a time, perform the calculation as

$$z_1 \times [1/(z_2 + z_3)].$$

Keystrokes	Display	
FIX 4		Sets the display format used here.
XEQ SIZE 005		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 005.
1 ENTER↑	1.0000	
2 CHS ENTER↑	−2.0000	
3 CHS ENTER↑ 4	4_	
XEQ C+	U=2.0000	Real part $(z_2 + z_3)$.
R/S	V=−2.0000	Imaginary part $(z_2 + z_3)$.
XEQ CINV	U=0.2500	$1/(z_2 + z_3)$
R/S	V=0.2500	
13 ENTER↑ 23	23_	
XEQ C×	U=2.5000	$z_1/(z_2 + z_3)$
R/S	V=9.0000	

Find the three cube roots of 8.

Keystrokes	Display
0 ENTER↑	0.0000
8 ENTER↑ 3	3_
XEQ Z↑1/N	U=2.0000
R/S	V=0.0000
R/S	U=−1.0000
R/S	V=1.7321
R/S	U=−1.0000
R/S	V=−1.7321

Evaluate $e^{z^{-2}}$, where $z = (1 + i)$.

Keystrokes	**Display**	
1 [ENTER↑]	1.0000	
1 [ENTER↑] 2	2_	
[XEQ] [Z↑N]	U=0.0000	z^2
[R/S]	V=2.0000	
[XEQ] [CINV]	U=0.0000	z^{-2}
[R/S]	V=−0.5000	
[XEQ] [e↑Z]	U=0.8776	$e^{z^{-2}}$
[R/S]	V=−0.4794	

Evaluate sin $(2 + 3i)$.

Keystrokes	**Display**
3 [ENTER↑] 2	2_
[XEQ] [SINZ]	U=9.1545
[R/S]	V=−4.1689

VECTOR OPERATIONS

The VC program simulates a "Vector Calculator" superimposed on your normal calculator. It redefines the functions in the top two rows of keys to these vector operations: addition, subtraction, distance, dot product, cross product, angle between vectors, norm, and unit vector. This pac also offers these operations to you as regular functions (*without* the Vector Calculator) that you can execute like any other HP–41 (nonkeyboard) function. Their Alpha names are given under "Summary of Vector Operations".

The vector operations operate on *three-dimensional* vectors described in *rectangular coordinates*. That is, every vector has three components, V_x, V_y, and V_z. For a two-dimensional vector, z must be equal to zero.

A complement to VC is the Coordinate Transformations program, TR. This means you can carry out vector operations and transformations on the same data, since you can access either program from the other one. The use of coordinate transformations is covered in the next chapter, "Coordinate Transformations".

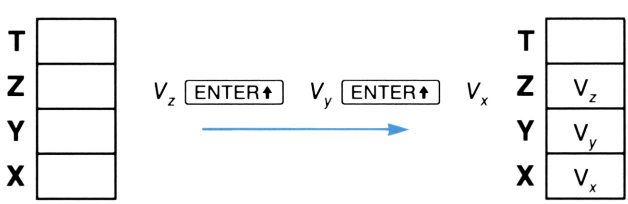
Method

The Vector Stack

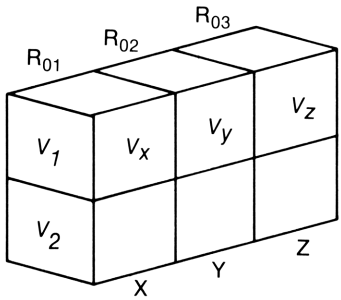
$\vec{V}_1$	x_1, y_1, z_1
$\vec{V}_2$	x_2, y_2, z_2

The Vector Calculator (program VC) creates a *vector stack* that works in concert with the regular RPN stack (X-, Y-, Z-, and T-registers). When you enter the three components of a vector in the order V_z, V_y, V_x, they occupy the regular stack like so:

The RPN Stack



How do the two stacks relate to each other? Basically, the *"bottom" level* of the vector stack (V_2) is stored in registers X, Y, and Z of the stack, while the *"upper" level* of the vector stack (V_1) is stored in data storage registers R_{01}, R_{02}, and R_{03}. You can imagine the registers shared in a three-dimensional stack like so:



The vector stack is two vector-levels high, so it accomodates two vectors. Note, however, that each level contains *three* components: the x-, y-, and z-components for each vector.

The diagram on the next page shows you what happens in vector entry and vector-stack movement from the point-of-view of the vector stack and from the point-of-view of the RPN and vector stacks together.

When you enter two vectors (as you would prior to executing a typical vector operation), the first one you key in becomes V_1 and the second one you key in becomes V_2. A "vector entry" (the function VE, or pressing R/S in the Vector Calculator) copies the bottom vector (V_2) into the top vector (V_1). Then, when you key in the next vector, it overwrites the copy in the bottom vector (V_2), leaving the first vector in V_1 and the second vector in V_2.

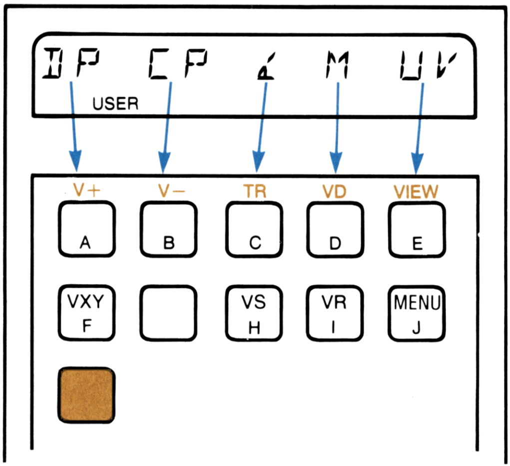
Vector-Stack Lift

	1. Enter vector's components:	2. Vector enter:	3. Enter second vector's components:
Vector Stack	V_1 V_2 x_1, y_1, z_1	V_1 x_1, y_1, z_1 V_2 x_1, y_1, z_1	V_1 x_1, y_1, z_1 V_2 x_2, y_2, z_2
Input	V_z ENTER↑ V_y ENTER↑ V_x	VE (*or* R/S in Vector Calculator)	V_z ENTER↑ V_y ENTER↑ V_x
Vector and RPN Stacks	V_2 x_1 y_1 z_1 X Y Z	R_{01} R_{02} R_{03} V_1 x_1 y_1 z_1 V_2 x_1 y_1 z_1 X Y Z	R_{01} R_{02} R_{03} V_1 x_1 y_1 z_1 V_2 x_2 y_2 z_2 X Y Z

All two-vector operations with a vector result place the resulting vector in both V_1 and V_2. This facilitates chained (subsequent) vector calculations. A vector-recall copies V_2 to V_1 then puts the recalled vector into V_2.

Instructions

- Starting VC (invoking the vector calculator) does *not* clear the vector stack, so you can still work with previously stored vectors.
- Be sure to give each vector three dimensions. If it has only two dimensions, then enter a zero for V_z.
- Enter the vector's dimensions as *rectangular coordinates*. If you have polar coordinates (magnitude and angle) for a two-dimensional vector, convert them using the function P→R (*polar to rectangular*).
- For those operations involving angles, the units will match the current angular mode setting (Degrees, Radians, or Grads).
- The view function (E) is very useful for reviewing the components of V_2 in the stack.
- V_1 refers to the "top" vector; the one in R_{01}, R_{02}, and R_{03}. V_2 refers to the "bottom" vector; the one in X, Y, and Z.



This menu will show you which key corresponds to which function in VC. Press [J] to recall this menu to the display at any time.

To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then recall the menu by pressing [J]. (However, you do not *need* to clear the program's display before performing calculations.)

The Vector Calculator provides two methods for entering a vector into the vector stack. The vector-enter function ([VE]) is analogous to the [ENTER↑] key. A shortcut method of vector entry is the [R/S] key. Whenever you enter the vector components from the keyboard when the menu was the last thing displayed before keying in the three components, pressing [R/S] will perform the same function as [VE]

The following table shows the keystrokes to execute vector operations on the Vector Calculator (program VC). For a definition of each operation, refer to the "Summary of Vector Operations" following the Instruction Table.

Instruction Table for VC

		Size: 004
Instructions	**Key In:**	**Display**
1. Start the program for the Vector Calculator, VC.	[XEQ] [VC] *	**DP CP ∡ M UV**
2. Enter the three components of your first vector (V_1). Separate two vectors with a "vector enter" after the first set of coordinates: execute [VE] or—only if the menu was the last thing displayed before you entered the first component—press [R/S].	z_1 [ENTER↑] y_1 [ENTER↑] x_1 [R/S]	z_1 y_1 **DP CP ∡ M UV**
3. Key in the second vector (V_2). *Do not press* [R/S].	z_2 [ENTER↑] y_2 [ENTER↑] x_2	z_2 y_2 x_2
4. Display the main menu (optional).	[J]	**DP CP ∡ M UV**
5. Execute a vector operation:		
■ Dot Product, $V_1 \cdot V_2$	[A] (**DP**)	**DOT**=*result*
■ Cross Product, $V_1 \times V_2$	[B] (**CP**) [R/S]† [R/S]†	**X**=*x result* **Y**=*y result* **Z**=*z result*
■ Angle between V_1 and V_2	[C] (∡)	∡=*result*
■ Norm (magnitude) of V_2 (This also puts the unit vector of V_2 in Y, Z, T.)	[D] (**M**)	**M**=*result*
■ Unit Vector of V_2 (This also puts the norm in the T-register.)	[E] (**UV**) [R/S]† [R/S]†	**X**=*x result* **Y**=*y result* **Z**=*z result*
■ Vector Add, $V_1 + V_2$	[■] [A] [R/S]† [R/S]†	**X**=*x result* **Y**=*y result* **Z**=*z result*
■ Vector Subtract, $V_1 - V_2$	[■] [B] [R/S]† [R/S]†	**X**=*x result* **Y**=*y result* **Z**=*z result*
■ Coordinate Transformations—refer to the "Coordinate Transformations" chapter for instructions. [USER] [■] [C] retrieves the Vector Calculator.	[■] [C] [USER] [■] [C]	**Z0,Y0,X0 ?** **DP CP ∡ M UV**
■ Distance between V_1 and V_2	[■] [D]	**d**=*result*

Instruction Table for VC (Continued)

Instructions	Key In:	Display
6. Restore the main menu after or between operations (optional).	[J] (or [R/S])	DP CP ⊿ M UV
7. To view the components of V_2, the vector in the stack:	■ [E] [R/S]† [R/S]†	X=*x-coordinate* Y=*y-coordinate* Z=*z-coordinate*
8. To exchange V_1 and V_2 (the vector components in R_{01}, R_{02}, and R_{03} switch with those in X, Y, and Z):	[F]	DP CP ⊿ M UV
9. To store V_2's components as vector-register *n* in R_{3n+1}, R_{3n+2}, and R_{3n+3} ($n \geqslant 0$):	*n* [H]	DP CP ⊿ M UV
10. To recall the contents of vector-register *n* into V_2 (X, Y, and Z), pushing V_2 into V_1:	*n* [I] [R/S]† [R/S]†	X=*x-coordinate* Y=*y-coordinate* Z=*z-coordinate*

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

† If you have a printer attached, the display automatically returns to the main menu after printing the result(s).

Remarks

You can eliminate the display of results on the Vector Calculator by setting flag 04. This lets you perform successive calculations more quickly by not having to step through the display of the results. You can still view the results when you want by pressing ■ [E].

This program uses local Alpha labels (as explained in the owner's manual for the HP-41) assigned to keys [A]–[F], [H]–[J], and ■ [A]–■ [E]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

Summary of Vector Operations

The vector operations are accessible in two different ways:

- By using the Vector Calculator and its redefined keys, as explained above.
- By directly executing a vector function using its Alpha name, like any other HP-41 nonkeyboard function.
- V_1 refers to the first (or "top") vector: the one in R_{01}, R_{02}, and R_{03}. V_2 refers to the second (or "bottom") vector: the one in X, Y, and Z.

The operations perform the same calculations regardless of how they are executed. These characteristics are given in the table below, along with their Alpha names and descriptions.* You *can* also execute these operations by Alpha name from inside the Vector Calculator, though it is usually more convenient to use the Vector Calculator's redefined keys.

When using vector operations without the Vector Calculator—that is, when using their Alpha names (as given below)—it is best if **USER** is *not* on (User keyboard inactive). This avoids conflicts between User-key assignments made by the Vector Calculator and Normal keyboard functions (such as [x≷y]).

Table of Vector Operations

Function	Effect
[CROSS] (*cross product*)	$V_1 \times V_2$. Returns the three-dimensional product into V_2 (in X, Y, Z). A copy goes into V_1. R_0 is not preserved. Vector Calculator also uses [B] (**CP**).
[DOT] (*dot product*)	$V_1 \bullet V_2$. Returns the scalar product into the X-register. (V_2 is destroyed; V_1 unaffected.) Vector Calculator also uses [A] (**DP**).
[TR] (*coordinate transformations*)	Calls up the Coordinate Transformations program, TR. Refer to the next chapter. Vector Calculator also uses ■ [C].

* The vector-viewing operation is available only in the Vector Calculator, as is the norm operation. However, the norm is also returned as part of the unit-vector operation.

Table of Vector Operations (Continued)

Function	Effect
[UV] (*unit vector*)	Converts V_2 (in X, Y, Z) into its unit vector, and returns the norm to the T-register. (V_1 is unaffected.) Vector Calculator also uses [E] (**UV**). Note: the unit vector of (0,0,0) is (0,0,1) with a norm of zero.
[V+] (*vector addition*)*	$V_1 + V_2$. Returns the sum into both V_1 and V_2. Vector Calculator also uses ■ [A].
[V−] (*vector subtraction*)*	$V_1 - V_2$. Returns the difference into both V_1 and V_2. Vector Calculator also uses ■ [B].
[V*] (*vector scalar multiplication*)	$V_2 * a$. Multiplies V_2 (in Y, Z, T) by a in X-register, and returns result to X, Y, and Z.
[V∡] (*vector angle*)	Returns the angle into the X-register. The angle is expressed in the current angular setting. V_1 and V_2 are not preserved; the unit vector of V_2 ends up in V_1. Vector Calculator also uses [C] (∡). Note: the vector (0,0,0) is assumed to have the same direction as (0,0,1).
[VD] (*vector distance*)	Returns the scalar distance between V_1 and V_2 into the X-register. Also returns the difference vector ($V_1 - V_2$) into V_1. V_2 is not preserved. Vector Calculator also uses ■ [D].
[VE] (*vector enter*)	Analogous to [ENTER↑]. Used to separate the entry of two vectors (V_1, then V_2) prior to executing an operation. (Vector entry copies the first vector from X, Y, Z into R_{01}, R_{02}, R_{03}.) In the Vector Calculator you can press [R/S] instead, but only if the menu was just displayed.
[VR] (*vector recall*)	With n ($n>0$)† in the X-register, copies V_2 to V_1, then recalls a three-dimensional vector from vector-register n into V_2 (X, Y, and Z) from storage registers R_{3n+1}, R_{3n+2}, and R_{3n+3}. Analogous to [RCL]. (The previous V_2 is lifted into V_1, overwriting V_1.) Vector Calculator also uses [I].

Table of Vector Operations (Continued)

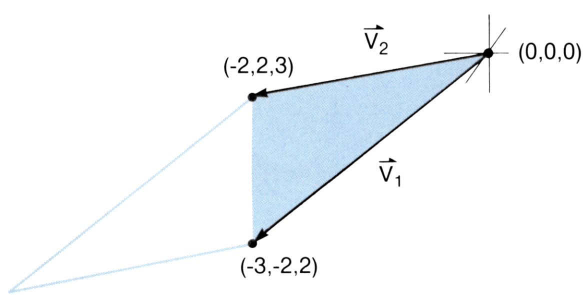
Function	Effect
VS (*vector store*)	With n ($n>0$)† in the X-register, copies and stores V_2 (now in Y, Z, and T) as vector-register n in storage registers R_{3n+1}, R_{3n+2}, and R_{3n+3}. Analogous to STO. (V_2 is unaffected.) Vector Calculator also uses H.
VXY (*vector exchange*)	V_1 exchanges values with V_2. Coordinates x_1, y_1, and z_1 move from R_{01}, R_{02}, and R_{03} into the X-, Y-, and Z-registers, while x_2, y_2, and z_2 move from X, Y, and Z into R_{01}, R_{02}, and R_{03}. Vector Calculator also uses F (x≷y).

* Remember that + and − are *shifted* Alpha characters.

† If $n = 0$ then VR and VS both copy V_2 to V_1, the same as VE. Do not use $n < 5$ if you plan to store vectors for use with the TR program (■ C).

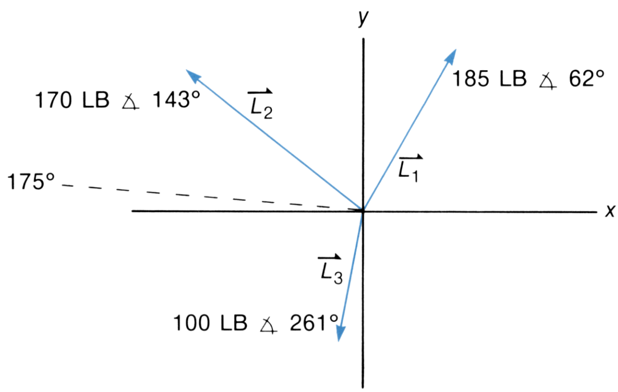
Examples

Find the area of the triangle determined by the vectors $V_1 = (-3,-2,2)$ and $V_2 = (-2,2,3)$. Recall that the area of the parallelogram determined by V_1 and V_2 equals the norm of $V_1 \times V_2$.



Keystrokes	Display	
[FIX] 4		Sets the display format used here.
[XEQ] [SIZE] 004		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 004.
[XEQ] [VC]	DP CP ∡ M UV	Starts the Vector Calculator. (You could also use the operations directly, without the Vector Calculator.)
2 [ENTER↑]	2.0000	Enter z_1, then y_1,
2 [CHS] [ENTER↑]	−2.0000	then key in x_1, ending
3 [CHS] [R/S]	DP CP ∡ M UV	with vector entry.
3 [ENTER↑]	3.0000	Enter z_2, then y_2,
2 [ENTER↑]	2.0000	then x_2.
2 [CHS]	−2_	
[J]	DP CP ∡ M UV	Retrieves the vector menu (optional).
[B] **(CP)**	X=−10.0000	
[R/S]	Y=5.0000	
[R/S]	Z=−10.0000	Result is (−10, 5, −10).
[J] or [R/S]	DP CP ∡ M UV	Ready to find norm.
[D] **(M)**	M=15.0000	Norm (magnitude), which equals the area of the parallelogram.
2 [÷]	7.5000	This is the area of the triangle, which is half that of the parallelogram.

Resolve the following three loads along a 175-degree line. Use the dot product on the sum of the three loads to do so. You will first need to convert the polar coordinates to rectangular coordinates. Remember to set $z = 0$.



Save the results for the polar coordinates of L_3 and the 175°-line so that you can re-use them to find the resolution (dot product) when L_3 is doubled. This example stores those results in vector-registers 1 and 2.

This solution uses Alpha (manual) execution of the vector operations, but you can use the Vector Calculator, as in the above example. Make sure that the User keyboard is not active.

Keystrokes	**Display**	
XEQ SIZE 010		Optional—sets the number of storage registers needed for this example (including vector storage). This is not necessary if your allocation is already SIZE ⩾ 010.
XEQ DEG *		Make sure the calculator is in Degrees mode.
0 ENTER↑	**0.0000**	Enters zero for the z-coordinate (in preparation for the vector operations *after* the coordinates are transformed).
62 ENTER↑	**62.0000**	To convert L_1 to rectangular coordinates, first enter θ, then key in r.
185 P→R	**86.8522**	x-coordinate for L_1.

* If the **USER** annunciator is on, press USER to turn it off.

Keystrokes	Display	
[x≶y]*	163.3453	y-coordinate for L_1. This step is optional—it lets you view y.
[x≶y]*	86.8522	Restores x to X and y to Y—*only necessary if you switched them* (in the last step).
[XEQ] [VE]	86.8522	No menu; displays previous result.
0 [ENTER↑]	0.0000	
143 [ENTER↑]	143.0000	
170 [P→R]	−135.7680	Displays x_2. L_2 is converted to rectangular coordinates.
[XEQ] [V+]	−48.9158	x-coordinate of resultant vector (in both V_1 and V_2).
0 [ENTER↑]	0.0000	
261 [ENTER↑]	261.0000	
100 [P→R]	−15.6434	x_3. L_3 is converted to rectangular coordinates.
1 [XEQ] [VS]	−15.6434	Stores L_3 in vector-register 1 (in R_4, R_5, R_6).
[XEQ] [V+]	−64.5592	x-coordinate of resultant vector of (L_1+ L_2+L_3) in both V_1 and V_2.
0 [ENTER↑]	0.0000	
175 [ENTER↑]	175.0000	
1 [P→R]	−0.9962	x-coordinate of the 175°-line.
2 [XEQ] [VS]	−0.9962	Stores 175°-line in vector-register 2 (in R_7, R_8, R_9).

* Note that when USER is on, you *cannot use* [x≶y] within the Vector Calculator to exchange X and Y because this key is redefined in the Vector Calculator to exchange V_1 and V_2. Use [R↓] instead.

Keystrokes	Display	
[XEQ] [DOT]	78.8586	The dot product is the resolution of the resultant L vector along the 175°-line.
[XEQ] [VXY]	−64.5592	Returns the resultant summed vector ($L_1+L_2+L_3$) to V_2 (X, Y, Z).
1 [XEQ] [VR]	−15.6434	Recalls L_3.
[XEQ] [V+]	−80.2027	Adds L_3 to the previous sum (in effect doubling L_3).
2 [XEQ] [VR]	−0.9962	Recalls the 175°-line.
[XEQ] [DOT]	85.8342	Finds the new dot product for the resolution of the new sum along the 175°-line.

Programming Information

The following subroutines in VC can be used in your own programs. They are three-dimensional vector operations for one or two vectors.

Minimum Size to Run: SIZE 004, not including vector-store and vector-recall.

Subroutines

Subroutine Name	Initial Registers	Final Registers
CROSS (*cross product*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} R_{01} = V_{1x} R_{02} = V_{1y} R_{03} = V_{1z}	X = $(V_1 \times V_2)_x$ Y = $(V_1 \times V_2)_y$ Z = $(V_1 \times V_2)_z$ R_{00} = scratch R_{01} = $(V_1 \times V_2)_x$ R_{02} = $(V_1 \times V_2)_y$ R_{03} = $(V_1 \times V_2)_z$
DP (*dot product*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} R_{01} = V_{1x} R_{02} = V_{1y} R_{03} = V_{1z}	X = $V_1 \cdot V_2$ R_{01} = V_{1x} R_{02} = V_{1y} R_{03} = V_{1z}

Subroutines (Continued)

Subroutine Name	Initial Registers	Final Registers
V+ (*vector add*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} $R_{01} = V_{1x}$ $R_{02} = V_{1y}$ $R_{03} = V_{1z}$	$X = V_{1x} + V_{2x}$ $Y = V_{1y} + V_{2y}$ $Z = V_{1z} + V_{2z}$ $R_{01} = V_{1x} + V_{2x}$ $R_{02} = V_{1y} + V_{2y}$ $R_{03} = V_{1z} + V_{2z}$
V− (*vector subtract*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} $R_{01} = V_{1x}$ $R_{02} = V_{1y}$ $R_{03} = V_{1z}$	$X = V_{1x} - V_{2x}$ $Y = V_{1y} - V_{2y}$ $Z = V_{1z} - V_{2z}$ $R_{01} = V_{1x} - V_{2x}$ $R_{02} = V_{1y} - V_{2y}$ $R_{03} = V_{1z} - V_{2z}$
V* (*vector scalar multiply*)	$X = a$ $Y = V_x$ $Z = V_y$ $T = V_z$	$X = V_x * a$ $Y = V_y * a$ $Z = V_z * a$
VA (*vector angle*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} $R_{01} = V_{1x}$ $R_{02} = V_{1y}$ $R_{03} = V_{1z}$	X = small angle between V_1 and V_2 R_{01} = unit vector V_{2x} R_{02} = unit vector V_{2y} R_{03} = unit vector V_{2z}
VD (*vector distance*)	X-register = V_{2x} Y-register = V_{2y} Z-register = V_{2z} $R_{01} = V_{1x}$ $R_{02} = V_{1y}$ $R_{03} = V_{1z}$	X = distance between V_1 and V_2 $R_{01} = V_{1x} - V_{2x}$ $R_{02} = V_{1y} - V_{2y}$ $R_{03} = V_{1z} - V_{2z}$
VE (*vector enter*)	X-register = V_x Y-register = V_y Z-register = V_z	X-register = V_x Y-register = V_y Z-register = V_z $R_{01} = V_x$ $R_{02} = V_y$ $R_{03} = V_z$
VR (*vector recall*)	$X = n$ $Y = V_x$ $Z = V_y$ $T = V_z$	$X = R_{3n+1}$ $Y = R_{3n+2}$ $Z = R_{3n+3}$ $R_{01} = V_x$ $R_{02} = V_y$ $R_{03} = V_z$

Subroutines (Continued)

Subroutine Name	Initial Registers	Final Registers
VS (*vector store*)	$X = n$ $Y = V_x$ $Z = V_y$ $T = V_z$	$X = V_x$ $Y = V_y$ $Z = V_z$ $R_{3n+1} = V_x$ $R_{3n+2} = V_y$ $R_{3n+3} = V_z$
VXY (*vector exchange*)	X-register $= V_{2x}$ Y-register $= V_{2y}$ Z-register $= V_{2z}$ $R_{01} = V_{1x}$ $R_{02} = V_{1y}$ $R_{03} = V_{1z}$	X-register $= V_{1x}$ Y-register $= V_{1y}$ Z-register $= V_{1z}$ $R_{01} = V_{2x}$ $R_{02} = V_{2y}$ $R_{03} = V_{2z}$
UV (*unit vector*)	$X = V_x$ $Y = V_y$ $Z = V_z$	X = unit vector *x* Y = unit vector *y* Z = unit vector *z* T = norm

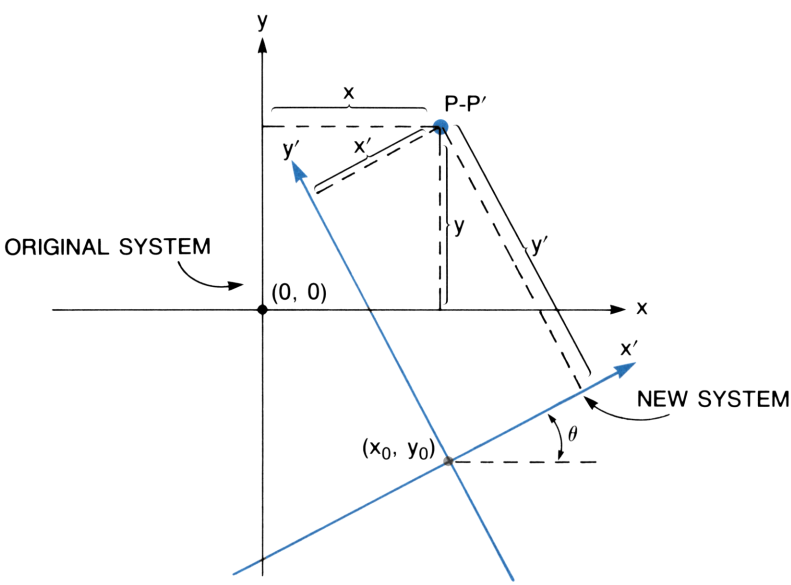
Comments. Vector operations work on one or two vectors. One is stored in the stack (X-, Y-, and Z-registers), another in R_{01}, R_{02}, and R_{03}. For a two-vector operation, V_1 is considered to be in R_{01}–R_{03} and V_2 is considered to be the vector in the stack. The vectors' components are stored in order; that is, V_x, V_y, and V_z into X, Y, and Z or into R_{01}, R_{02}, and R_{03}, respectively.

COORDINATE TRANSFORMATIONS

The TR program performs three-dimensional translation of coordinates, with or without rotation. This program uses parts of the VC program for vector operations. You can access TR either directly or from VC. (VC and the Vector Calculator are discussed in the preceding chapter, "Vector Operations".)

The program prompts you for the coordinates of the origin of the *new* system (x_0, y_0, z_0), the angle of rotation of this system relative to the *original* system, and the axis about which the rotation is performed. You can then enter points in the original system (x, y, z) that you want transformed to the new system (x', y', z'), or enter points in the new system (x',y', z') that you want transformed to the original system (x, y, z). For a two-dimensional case, enter z_0 as zero.

A Two-Dimensional Rotation about the Axis (0, 0, 1)



After specifying the new origin (x, y, z), you specify the rotation angle. For a three-dimensional system with a non-zero angle of rotation, you also specify its *rotation vector* (a, b, c). The rotation vector defines the axis about which the rotation is to be done; it can have any non-zero magnitude.

Equations

$$\vec{P}' = [(\vec{P} - \vec{T}) \cdot \vec{n}]\vec{n}(1 - \cos\theta) + (\vec{P} - \vec{T})\cos\theta + [(\vec{P} - \vec{T}) \times \vec{n}]\sin\theta$$

$$\vec{P} = [(\vec{P}' \cdot \vec{n})\vec{n}(1 - \cos\theta) + \vec{P}'\cos\theta + (\vec{P}' \times \vec{n})\sin(-\theta)] + \vec{T}$$

where

$\vec{P}'$ = new system coordinates
$\vec{P}$ = old system coordinates
$\vec{T}$ = origin of new system
$\vec{n}$ = unit rotation vector (a, b, c)
θ = rotation angle

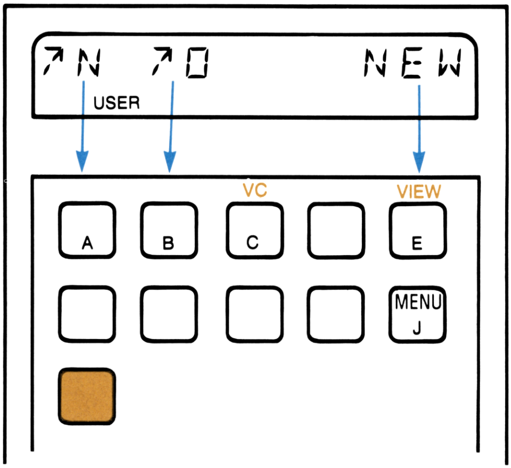
Two-dimensional transformations are handled as a special case of three-dimensional transformations with (a, b, c) set to (0, 0, 1).

Instructions

You can start **TR** either directly ([XEQ] [TR]) or from the Vector Calculator ([■] [C]) in **VC**. The Vector Calculator is covered in the "Vector Operations" chapter.

Enter coordinates as rectangular coordinates and specify angles according to the current setting (Degrees, Radians, or Grads mode).

- For two dimensions, input zero for the z-value.
- For pure translation, input zero for the rotation angle.
- For pure rotation, input zeros for x_0, y_0, and z_0.
- The sign of the rotation angle is determined by the right-hand rule and the direction of the rotation vector. For two dimensions, counter-clockwise rotation is considered positive.
- You can switch into and out of the Vector Calculator by pressing [■] [C]. ("C" for **C**alculator and **C**oordinate transformations). You can then perform vector operations upon vector coordinates in the stack and in storage registers. (Refer to "Remarks" for the storage locations of the vector coordinates.)
- The view function ([■] [E]) is very useful for reviewing the coordinates of the point in the stack.



Once you have entered your variables, this menu shows you which key corresponds to which function in TR. To restore this menu to the display at any time, press [J] *if the* **USER** *annunciator is on*. (If it is not on, press [USER] to turn it on.) Or, if the calculator is displaying results, you can press [R/S] until the menu appears. This will not disturb the program in any way.

To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then continue the program by pressing [J]. (However, you do not *need* to clear the program's display before performing calculations.)

Instruction Table for TR

		Size: 017
Instructions	**Key In:**	**Display**
1. Start program TR. The menu items in the display indicate the locations of functions in the top row of keys.	[XEQ] [TR]*	**Z0,Y0,X0 ?**
2. Enter the origin for the new system.	z_0 [ENTER↑] y_0 [ENTER↑] x_0 [R/S]	z_0 y_0 **ROT∡?**
3. Input the rotation angle of the new system:	θ [R/S]	**c,b,a ?**

Instruction Table for TR (Continued)

Instructions	Key In:	Display
4. For a three-dimensional system: Input the rotation vector's coordinates. For a two-dimensional system: just press [R/S].	*c* [ENTER↑] *b* [ENTER↑] *a* [R/S]	*c* *b* **↑N ↑O NEW**
5. To transform the coordinates of a point **from** the original system **to** the new system (**↑N**), enter the three coordinates of that point and select **↑N**. (For two dimensions, set $z=0$.)	*z* [ENTER↑] *y* [ENTER↑] *x* [A] (**↑N**) [R/S]† [R/S]† [R/S]†	*z* *y* **X**= *x′* **Y**= *y′* **Z**= *z′* **↑N ↑O NEW**
6. To transform the coordinates of a point **from** the new system **to** the original system (**↑O**), enter the three coordinates of that point and select **↑O**. (For two dimensions, set $z=0$.)	*z′* [ENTER↑] *y′* [ENTER↑] *x′* [B] (**↑O**) [R/S]† [R/S]† [R/S]†	*z′* *y′* **X**= *x* **Y**= *y* **Z**= *z* **↑N ↑O NEW**
7. To view the coordinates of the point in the stack:	■ [E] [R/S]† [R/S]† [R/S]†	**X**=*x-coordinate* **Y**=*y-coordinate* **Z**=*z-coordinate* **↑N ↑O NEW**
8. To transform another set of coordinates, go back to step 5 or 6.		
9. To set up a new transformed system, select **NEW** and then return to step 2.	[E] (**NEW**)	**Z0,Y0,X0 ?**
10. To use vector operations, switch to the Vector Calculator. All the functions described in the "Vector Operations" chapter are then available to you.	■ [C] (**USER** must be on)	**DP CP ∡ M UV**
11. To return to the TR program from VC:	■ [C]	**Z0,Y0,X0 ?**
12. To transform a vector result V_2 from VC, bypass the initial prompts and call up the main menu (assuming a transformed system is already defined):	[USER] [J]	**↑N ↑O NEW**

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

† This keystroke is unnecessary if you have a printer attached because the printer automatically prints the results and then displays the selection menu.

Remarks

This program uses local Alpha labels (as explained in the owner's manual for the HP–41) assigned to keys [A], [B], [E], ■[C], and [J]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

However, these local Alpha labels are active only while the **USER** annunciator is on. This allows you to use the arithmetic functions in the top two rows while the **USER** annunciator is off. (As long as **USER** is on, the keys mentioned above are redefined and will not execute their Normal functions.)

Data Storage. The vector or point you want to transform is stored in R_{04}, R_{05}, R_{06}, which is vector-storage register 1 (initially from the X-, Y-, and Z-registers). The rotation vector is stored in R_{07}, R_{08}, R_{09}, which is vector-storage register 2. The origin of the new system is stored in R_{10}, R_{11}, R_{12}, which is vector-storage register 3. The rotation angle is stored in R_{16}, while R_{13}, R_{14}, and R_{15} are used for scratch.

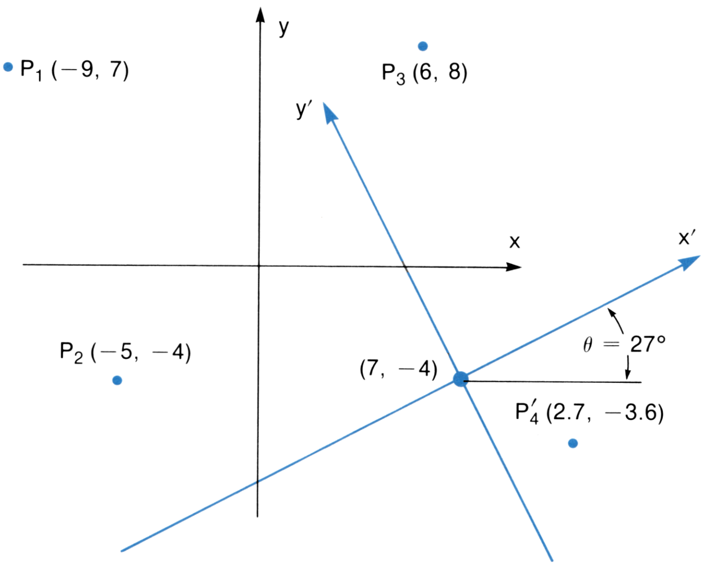
If you will be using vector storage operations ([VS], [VR], and the Vector Calculator) along with TR, keep in mind that TR uses R_0–R_{16} when it is initialized ([XEQ] [TR]). This means you should not store vectors in vector registers 1 through 5 (if you plan to use TR in your vector calculations).

Flags. Flag 01 is used to indicate whether the transformation is to be made to the new system or to the original system. When flag 1 is set, the transformation is to the new system.

Flag 05 is set when the system is rotated.

Examples

The coordinate systems (x, y) and (x', y') are shown below.



Convert the points P_1, P_2, and P_3 to equivalent coordinates in the (x', y') system. Convert the point P_4' to equivalent coordinates in the (x, y) system.

Keystrokes	Display	
FIX 4		Sets the display format used here.
XEQ SIZE 017		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 017.
XEQ TR	**Z0,Y0,X0 ?**	Prompts for z_0, y_0, and x_0 of new system.
0 ENTER↑	**0.0000**	Enters zero for z_0.
4 CHS ENTER↑	**−4.0000**	
7 R/S	**ROT∆?**	Prompts for angle of rotation.

Keystrokes	**Display**	
27 [R/S]	**c,b,a ?**	Prompts for the rotation vector. Skip this for a two-dimensional system.
[R/S]	**↑N ↑O NEW**	Prompts for P_1.
0 [ENTER↑] 7 [ENTER↑]	**7.0000**	
9 [CHS] [A] (**↑N**)	**X=−9.2622**	x_1'
[R/S]	**Y=17.0649**	y_1'
[R/S]	**Z=0.0000**	z_1'
[R/S]	**↑N ↑O NEW**	Ready for P_2. This step is optional—it brings up the main menu.
0 [ENTER↑] 4 [CHS] [ENTER↑]	**−4.0000**	
5 [CHS] [A] (**↑N**)	**X=−10.6921**	x_2' from P_2.
[R/S]	**Y=5.4479**	y_2'
[R/S]	**Z=0.0000**	z_2'
[R/S]	**↑N ↑O NEW**	Brings back the menu for your review.
0 [ENTER↑] 8 [ENTER↑]	**8.0000**	
6 [A] (**↑N**)	**X=4.5569**	x_3' from P_3.
[R/S]	**Y=11.1461**	y_3'
[R/S]	**Z=0.0000**	z_3'
[R/S]	**↑N ↑O NEW**	Brings back the menu for your review.
0 [ENTER↑] 3.6 [CHS] [ENTER↑]	**−3.6000**	
2.7 [B] (**↑O**)	**X=11.0401**	x_4 from P_4'.
[R/S]	**Y=−5.9818**	y_4
[R/S]	**Z=0.0000**	z_4

A three-dimensional coordinate system is translated to (2.45, 4.00, 4.25). After the translation, a 62.5 degree rotation occurs about the (0, −1, −1) axis. In the original system, a point had the coordinates (3.9, 2.1, 7.0). What are the coordinates of the point in the translated, rotated sytem?

Keystrokes	Display	
[J]	↑N ↑O NEW	Retrieves menu (if **USER** is on).
[E] (**NEW**)	Z0,Y0,X0 ?	Prompts for a new system.
4.25 [ENTER↑] 4 [ENTER↑]	4.0000	
2.45 [R/S]	ROT∡?	
62.5 [R/S]	c,b,a ?	
1 [CHS] [ENTER↑]	−1.0000	
1 [CHS] [ENTER↑]	−1.0000	
0 [R/S]	↑N ↑O NEW	Ready for *P*.
7 [ENTER↑] 2.1 [ENTER↑]	2.1000	
3.9 [A] (↑**N**)	X=3.5861	x'
[R/S]	Y=0.2609	y'
[R/S]	Z=0.5891	z'

In the translated, rotated system above, a point has the coordinate (1, 1, 1). What are the corresponding coordinates in the original system?

Keystrokes	Display	
[R/S]	↑N ↑O NEW	Retrieves main menu. Optional step.
1 [ENTER↑] 1 [ENTER↑]	1.0000	
1 [B] (↑**O**)	X=2.9117	x
[R/S]	Y=4.3728	y
[R/S]	Z=5.8772	z

Programming Information

The subroutine CT can be used in your own programs. It performs coordinate transformations (rotations and translations) in three dimensions. It takes the x-, y-, and z-values from the stack (X-, Y-, and Z-registers) and transforms them to another system, or from the new system to the original system.

Minimum Size to Run CT: SIZE 017.

Flags Used: 01, 05.

Subroutine: CT

Initial Registers	Final Registers	Flags to Initialize
X-register = *x*-coordinate	X-register = transformed *x*-coordinate	SF 01 to transform to the new system
Y-register = *y*-coordinate	Y-register = transformed *y*-coordinate	CF 01 to transform to the original system
Z-register = *z*-coordinate	Z-register = transformed *z*-coordinate	SF 05 to rotate the coordinates
	$R_{00} = (1 - \cos\theta)(N \cdot P)$	CF 05 to not rotate the coordinates
	R_{01} = contents of X-register	
	R_{02} = contents of Y-register	
	R_{03} = contents of Z-register	
	$R_{04} = P_x$ (or $P_x - T_x$ if flag 01 set)	
	$R_{05} = P_y$ (or $P_y - T_y$ if flag 01 set)	
	$R_{06} = P_z$ (or $P_z - T_z$ if flag 01 set)	
$R_{07} = a$ (N_x, the unit rotation vector)	$R_{07} = a$ (N_x, the unit rotation vector)	
$R_{08} = b$ (N_y)	$R_{08} = b$ (N_y)	
$R_{09} = c$ (N_z)	$R_{09} = c$ (N_z)	
$R_{10} = T_x$, the translation vector	$R_{10} = T_x$, the translation vector	
$R_{11} = T_y$	$R_{11} = T_y$	
$R_{12} = T_z$	$R_{12} = T_z$	
R_{16} = rotation angle	R_{16} = rotation angle	

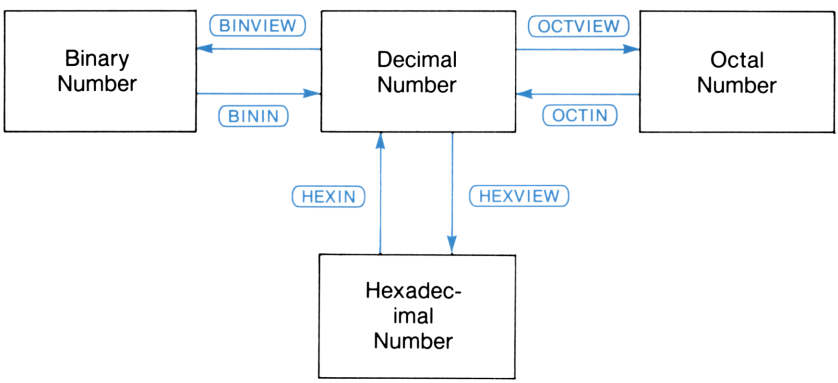
Comments. To use CT, load the translation vector (T), the unit rotation vector (N), and the rotation angle, set flag 01 to go to the new system or clear flag 01 to go to the original system. Set flag 05 to rotate the vector's coordinates (P). The result is returned to the X-, Y-, and Z-registers and in R_{01}, R_{02}, and R_{03}.

NUMBER CONVERSIONS AND BOOLEAN LOGIC

This pac includes several functions for calculating and manipulating binary, octal, and hexadecimal numbers. There are six functions for number conversion, four Boolean functions, and two bit-manipulating functions. All functions use a word length of 32 bits.

Number Conversion Functions

Six functions are provided for converting numbers between decimal values and the equivalent binary, octal, and hexadecimal values. The figure below illustrates the action of these six functions.



Valid Input Range for Data

- The binary input for BININ must be 0's and 1's; ten digits maximum.
- The decimal input for BINVIEW must be an integer from 0 through 1,023. Non-integers are truncated. The absolute value is used.
- The octal input for OCTIN must be digits from 0 through 7; ten digits maximum.

- The decimal input for [OCTVIEW] must be an integer from 0 through 1,073,741,823. Non-integers are truncated. The absolute value is used.
- The hexadecimal input for [HEXIN] must be digits from 0 through 9 and "letters" A through F; eight digits maximum.
- The decimal input for [HEXVIEW] must be an integer from 0 through 4,294,967,295. Non-integers are truncated. The absolute value is used.

Instructions

- The "VIEW" functions convert the *display* of the (decimal) value in the X-register. (The stack continues to hold the decimal version.) Press [←] to display the X-register again.
- The current [FIX] format determines the number of digits displayed between commas of the non-decimal number.
- The "IN" functions are *prefix* functions: *first* you execute the function, *then* you key in your value. Press [ENTER] to see the result.
- To abort an "IN" function press [ALPHA] [ALPHA].
- An "IN" function executed in a program will halt that running program.

Number Conversion Functions

Function	Effect
[BININ] (*binary to decimal*)	Converts a binary input to a decimal value in the X-register. 1. Execute [BININ]. The display shows _ **B**. 2. Input a binary number. 3. Press [ENTER↑] for result.
[BINVIEW] (*decimal to binary*)	Temporarily displays the binary equivalent of the decimal value in the X-register. 1. Input decimal value to convert. 2. Execute [BINVIEW]. 3. Displays *result* **B**. 4. Press [←] to see X-register again.

Number Conversion Functions (Continued)

Function	Effect
[OCTIN] (*octal to decimal*)	Converts an octal input to a decimal value in the X-register. 1. Execute [OCTIN]. The display shows _ **O**. 2. Input an octal number. 3. Press [ENTER↑] for result.
[OCTVIEW] (*decimal to octal*)	Temporarily displays the octal equivalent of the decimal value in the X-register. 1. Input decimal value to convert. 2. Execute [OCTVIEW]. 3. Displays *result* **O**. 4. Press [←] to see X-register again.
[HEXIN] (*hexadecimal to decimal*)	Converts a hexadecimal input to a decimal value in the X-register. 1. Execute [HEXIN]. The display shows _ **H**. 2. Input a hexadecimal number. 3. Press [ENTER↑] for result.
[HEXVIEW] (*decimal to hexadecimal*)	Temporarily displays the hexadecimal equivalent of the decimal value in the X-register. 1. Input decimal value to convert. 2. Execute [HEXVIEW]. 3. Displays *result* **H**. 4. Press [←] to see X-register again.

Boolean Functions

Included in this group of functions are Boolean logic, bit checking, and bit rotation.

Valid Input Range for Data

These functions operate on decimal numbers in the range zero through 4,294,967,295 (32-bit, unsigned integers). Non-integers are truncated. For negative values, the absolute value is used.

Instructions

The result of a Boolean operation is returned to the X-register. The original value of the X-register is saved in the LAST X register *except* for [BIT?], which does not affect LAST X or the stack. All other two-parameter functions drop the stack.

Boolean Functions

Function	Effect
AND	Calculates the logical AND of x and y.
OR	Calculates the logical inclusive OR of x and y.
XOR	Calculates the logical exclusive OR of x and y.
NOT	Takes the one's complement of $\lvert x \rvert$.
BIT? (*test bit*)	Tests the bit in the Y-register specified by the value in the X-register. If the bit is one, the calculator displays **YES**; if the bit is zero, the calculator displays **NO**. In a program, BIT? is a conditional function following the "do if true" rule: a one bit causes the next program step to be executed, while a zero bit causes the next program step to be skipped.
ROTXY (*rotate Y by X*)	Rotates the value in the Y-register to the right by the number of bits specified in the X-register. Rotating right $(32-x)$ bits is equivalent to rotating left x bits.

Examples

What are the binary, octal, and hexadecimal equivalents of 65_{10}? Set FIX 4 so that commas separate every four digits.

Keystrokes	Display	
FIX 4		Sets the display format used here.
65	**65_**	
XEQ BINVIEW	**100,0001 B**	Binary.
XEQ OCTVIEW	**101 O**	Octal.
XEQ HEXVIEW	**41 H**	Hexadecimal.

What is the octal result of rotating $FA407_{16}$ six bits to the right, adding 100100_2, and then ANDing the result with 25_{10}?

Keystrokes	Display	
XEQ HEXIN	_ H	
FA407	FA407_ H	
ENTER↑	1,025,031.000	Decimal equivalent of $FA407_{16}$.
6	6_	
XEQ ROTXY	469,778,064.0	Rotates value right six bits.
XEQ BININ	_ B	
100100	10,0100_ B	
+	469,778,100.0	Adds binary entry to previous value.
25	25_	
XEQ AND	16.0000	ANDs 25 with previous result.
XEQ OCTVIEW	20 O	Octal result.

CURVE FITTING

The CFIT program collects and fits statistical data (x_i, y_i) to one of the following four chosen curves or to the curve of best fit. The curve of best fit is considered to be the one with the highest coefficient of determination, r^2, for the data.

- Straight line (linear regression), $y = a + bx$
- Exponential curve, $y = ae^{bx}$ (where $a > 0$)
- Logarithmic curve, $y = a + b(\ln x)$
- Power curve, $y = ax^b$ (where $a > 0$)

The program solves for a, b, r^2, and $\hat{y}$, the linear estimate (a predicted value for y).

Equations

The regression coefficients a and b are found by solving the following linear equations, where n is the total number of data pairs.

$$An + b\Sigma X_i = \Sigma Y_i$$

$$A\Sigma X_i + b\Sigma(X_i)^2 = \Sigma(Y_i X_i)$$

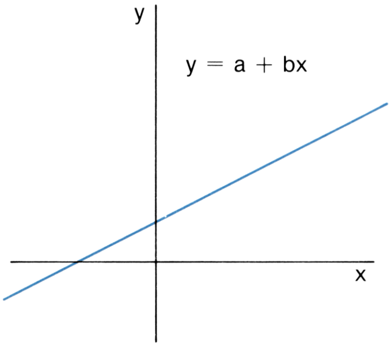
Definitions of Regression Variables

Regression	A	X_i	Y_i
Linear	a	x_i	y_i
Exponential	$\ln a$	x_i	$\ln y_i$
Logarithmic	a	$\ln x_i$	y_i
Power	$\ln a$	$\ln x_i$	$\ln y_i$

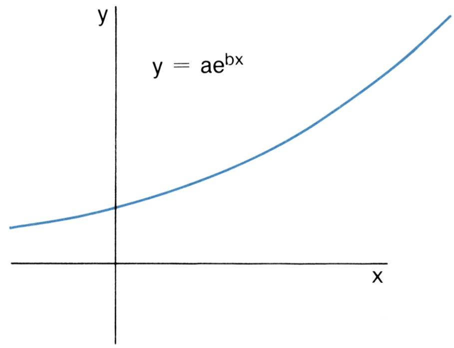
The coefficient of determination is

$$r^2 = \frac{A\Sigma Y_i + b\Sigma(X_i Y_i) - \frac{1}{n}(\Sigma Y_i)^2}{\Sigma(Y_i)^2 - \frac{1}{n}(\Sigma Y_i)^2}$$

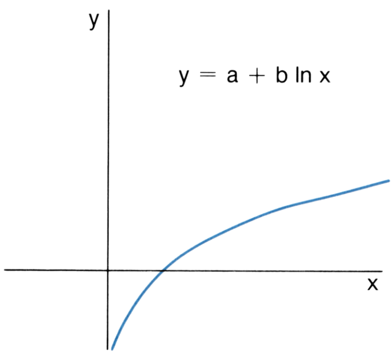
Linear Regression



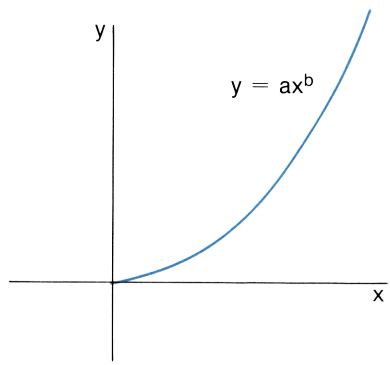
Exponential Curve Fit



Logarithmic Curve Fit



Power Curve Fit



Valid Input Range for Data

Program **CFIT** evaluates the given data by the least-squares method, using either the original equation (straight line and logarithmic curve) or the transformed equations (exponential curve and power curve).

All data values (x_i, y_i) must be positive and non-zero, otherwise **DATA ERROR** results.

As the difference between x-values and y-values becomes small, the accuracy of the regression coefficients decreases.

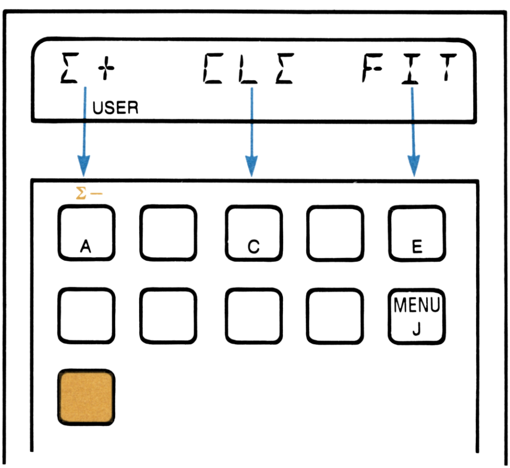
Note also that inaccurate results can be generated if one variable is much larger than the other or changes much more rapidly than the other does. (This occurs when the calculator would have to maintain more than ten significant digits for accuracy, which it can't.) If your data values are like this, you should apply scaling methods to maintain the accuracy of the results. Scaling methods are described in many statistics texts.

A **DATA ERROR** will result if you try to fit a curve containing only one data point, or if you use negative or zero data.

Instructions

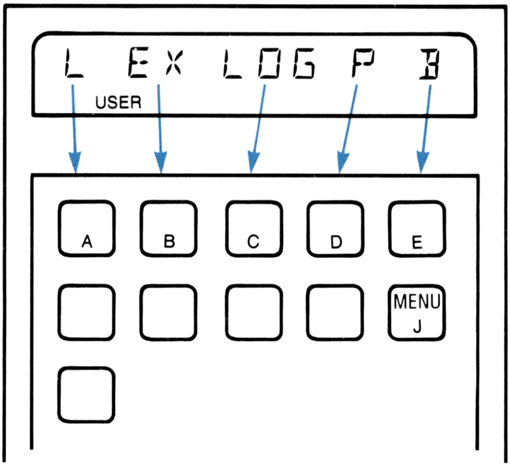
- The CFIT program starts with its home menu, Σ+ **CLΣ** **FIT**. This is for entering your statistical data: Σ+ to enter (y first, then x), ■ Σ+ to delete, and **CLΣ** to clear old statistical data. **FIT** brings up the curves menu.
- The curves menu, **L EX LOG P B**, offers you a choice of curves to which to fit your data: Linear, EXponential, LOGarithmic, Power, and Best fit. The best fit picks the curve that best fits your data.
- Once you've picked the curve to fit, pressing [R/S] displays successive regression variables. Pressing [J] brings back the home menu.

Home Menu



This menu will show you which key corresponds to which function in CFIT. Press [J] to recall this menu to the display at any time. This will not disturb the program in any way.

Curves Menu



To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then recall the home menu by pressing [J]. (However, you do not *need* to clear the program's display before performing calculations.)

Instruction Table for CFIT

Instructions	Key In:	Display
		Size: 018
1. Start program CFIT. The menu items in the display indicate the locations of functions in the top row of keys.	[XEQ] [CFIT]*	Σ+ CLΣ FIT
2. Clear old statistical data. (This is not necessary if you've just executed [CFIT], which automatically clears old data, too.)	[C] (CLΣ)	Σ+ CLΣ FIT
3. Input your data pairs. Repeat for each pair.	*y* [ENTER↑] *x* [A] (Σ+)†	*y* Σ+ CLΣ FIT
4. To see how many data pairs you have entered so far, clear the display (optional).	[←] [J]	*n* Σ+ CLΣ FIT

Instruction Table for CFIT (Continued)

Instructions	Key In:	Display
5. To correct any data pair, first re-enter that pair to delete it.	y_k [ENTER↑] x_k ■ [A] ([Σ−])	y_k Σ+ CLΣ FIT
Then enter the correct pair. (Step 3.)		
6. Display the curves menu.	[E] (FIT)	L EX LOG P B
7. Select the curve you want to fit.	[A] (L) [B] (EX) [C] (LOG) [D] (P) [E] (B)	LIN EXP LOG POW (the "best fit" of the above)
8. Find the values for *a*, for *b*, for r^2.	[R/S]‡ [R/S]‡ [R/S]‡	a= *result* b= *result* R↑2= *result*
9. Find the linear estimate, $\hat{y}$. Repeat as desired.	[R/S]‡ *x* [R/S] [R/S]‡	X=? Y=*result* X=?
10. To start over (recall the home menu):	[J]	Σ+ CLΣ FIT

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

† With a printer attached this step can give you a print-out of the values you just entered. Refer to your printer's owner's manual for instructions.

‡ This keystroke is unnecessary if you have a printer attached because the printer automatically prints the results and then displays the selection menu.

Remarks

This program uses local Alpha labels (as explained in the owner's manual for the HP-41) assigned to keys [A]–[E], ■ [A], and [J]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

Note: The CFIT program changes the location of the statistical registers. If you want to access information in the statistical registers *after using this program*, you must re-establish these registers in a known location using the function [ΣREG] (refer to the HP-41 owner's manual). This is true even if you just want to have the statistical registers in their default locations, R_{11}–R_{16}. To access statistical information stored *by* this program, refer to "Programming Information" at the end of this chapter.

Examples

Fit a straight line to the following set of data and compute $\hat{y}$ for $x = 37$ and $x = 35$.

x	40.5	38.6	37.9	36.2	35.1	34.6
y	104.5	102	100	97.5	95.5	94

Keystrokes	Display	
[FIX] 4		Sets the display format used here.
[XEQ] [SIZE] 018		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 018.
[XEQ] [CFIT]	**Σ+ CLΣ FIT**	Starting the program also clears old statistical data.
104.5 [ENTER↑] 40.5	**40.5_**	Enter first data pair, y first.
[A] (Σ+)	**Σ+ CLΣ FIT**	
102 [ENTER↑] 38.6	**38.6_**	Second pair.
[A] (Σ+)	**Σ+ CLΣ FIT**	
100 [ENTER↑] 37.9	**37.9_**	And so on.
[A] (Σ+)	**Σ+ CLΣ FIT**	
97.5 [ENTER↑] 36.2	**36.2_**	
[A] (Σ+)	**Σ+ CLΣ FIT**	
95.5 [ENTER↑] 35.2	**35.2_**	Oops! Wrong entry for x.
[A] (Σ+)	**Σ+ CLΣ FIT**	
95.5 [ENTER↑] 35.2	**35.2_**	Delete incorrect pair.
[■] [A] ([Σ−])	**Σ+ CLΣ FIT**	

Keystrokes	Display	
95.5 [ENTER↑] 35.1	35.1_	Enter correct pair.
[A] ($\Sigma+$)	$\Sigma+$ CLΣ FIT	
94 [ENTER↑] 34.6	34.6_	
[A] ($\Sigma+$)	$\Sigma+$ CLΣ FIT	
[E] (**FIT**)	L EX LOG P B	The curves menu.
[A] (**L**)	LIN	Selects the linear curve.
[R/S]	a=33.5271	
[R/S]	b= 1.7601	
[R/S]	R↑2=0.9909	
[R/S]	X=?	Asks for x-value for which you'd like to estimate y.
37 [R/S]	Y=98.6526	$\hat{y}$.
[R/S]	X=?	
35 [R/S]	Y=95.1323	
[R/S]	X=?	
[R/S]	L EX LOG P B	Returns the curves menu, ready to fit another curve to the data.
[R/S]	$\Sigma+$ CLΣ FIT	Returns the home menu, ready to start a new problem.

Enter the following set of data and find the best curve to fit it. Then compute $\hat{y}$ for $x = 1.5$ and $x = 2$.

x	0.72	1.31	1.95	2.58	3.14
y	2.16	1.61	1.16	0.85	0.5

Keystrokes	Display	
[R/S]	$\Sigma+$ CLΣ FIT	Make sure the home menu is displayed.
[C] (**CLΣ**)	$\Sigma+$ CLΣ FIT	Clears data from first example.
2.16 [ENTER↑] .72	.72_	Enters first data pair.
[A] ($\Sigma+$)	$\Sigma+$ CLΣ FIT	
1.61 [ENTER↑] 1.31	1.31_	
[A] ($\Sigma+$)	$\Sigma+$ CLΣ FIT	
1.16 [ENTER↑] 1.95	1.95_	
[A] ($\Sigma+$)	$\Sigma+$ CLΣ FIT	

Keystrokes	Display	
.85 [ENTER↑] 2.58	2.58_	
[A] (Σ+)	Σ+ CLΣ FIT	
.5 [ENTER↑] 3.14	3.14_	
[A] (Σ+)	Σ+ CLΣ FIT	
[E] (**FIT**)	L EX LOG P B	
[E] (**B**)	LOG	The best curve to fit is a logarithmic one.
[R/S]	a= 1.8515	
[R/S]	b=−1.1021	
[R/S]	R↑2=0.9893	
[R/S]	X=?	
1.5 [R/S]	Y=1.4046	
[R/S]	X=?	
2 [R/S]	Y=1.0875	
[R/S]	X=?	
[R/S]	L EX LOG P B	

Programming Information

The subroutines AΣ, DΣ, FIT, and BFIT can be used in your own programs.

- AΣ adds the data pair in the X- and Y-registers to a statistical register set to obtain summary statistics.
- DΣ deletes the data pair in the X- and Y-registers from the statistical register set.
- FIT fits a curve of type 1 through 4 to statistical data stored by the program CFIT or subroutines AΣ and DΣ.
- BFIT finds the best-fit curve of type 1 through 4 to statistical data stored by the program CFIT or subroutines AΣ and DΣ.

Minimum Size to Run: SIZE 018

Flags Used: BFIT and FIT use 01, 02, 03, 04, 06, 07.
AΣ and DΣ do not use any flags.

Subroutines: AΣ and DΣ

Initial Registers	Final Registers	Flags to Initialize
Y-register: *y*-value		
X-register: *x*-value		
$R_{04} = 0$	$R_{04} = \Sigma(y \ln x)$	
$R_{05} = 0$	$R_{05} = \Sigma(x \ln y)$	
$R_{06} = 0$	$R_{06} = \Sigma y$	
$R_{07} = 0$	$R_{07} = \Sigma y^2$	
$R_{08} = 0$	$R_{08} = \Sigma x$	
$R_{09} = 0$	$R_{09} = \Sigma x^2$	
$R_{10} = 0$	$R_{10} = \Sigma(xy)$	
$R_{11} = 0$	$R_{11} = n$	
$R_{12} = 0$	$R_{12} = \Sigma(\ln y)$	
$R_{13} = 0$	$R_{13} = \Sigma(\ln y)^2$	
$R_{14} = 0$	$R_{14} = \Sigma(\ln x)$	
$R_{15} = 0$	$R_{15} = \Sigma(\ln x)^2$	
$R_{16} = 0$	$R_{16} = \Sigma(\ln x)(\ln y)$	
$R_{17} = 0$	$R_{17} = n$, and temporarily Σy	

Subroutine: FIT

Initial Registers	Final Registers	Flags to Initialize
X-register = 1 = linear 2 = exponential 3 = logarithmic 4 = power		CF 01
	$R_{00} = 1, 2, 3,$ or 4	CF 02
	$R_{01} = a$	CF 03
	$R_{02} = b$	CF 04
	$R_{03} = r^2$	
R_{04}–R_{17}: all statistical registers are the same as above in AΣ.		

Comments. After loading the statistical registers using AΣ and DΣ, put the number of the curve to fit (1, 2, 3, 4) in the X-register and execute FIT. FIT sets flag 07 and sets a flag (01–04) that matches the curve type. It stores a, b, and r^2 in R_{01}, R_{02}, and R_{03}.

Subroutine: BFIT

<table>
<tr><th>Initial Registers</th><th>Final Registers</th><th>Flags to Initialize</th></tr>
<tr><td></td><td>R_{00} = 1, 2, 3, or 4
$R_{01} = a$
$R_{02} = b$
$R_{03} = r^2$</td><td rowspan="2">CF 01
CF 02
CF 03
CF 04</td></tr>
<tr><td colspan="2">R_{04}–R_{17}: all statistical registers are the same as above in AΣ and FIT.</td></tr>
</table>

Comments. After loading the statistical registers using AΣ and DΣ, execute BFIT and it will find the best fit of a linear, exponential, log, or power curve. BFIT sets flag 01 (linear curve), 02 (exponential curve), 03 (logarithmic curve), or 04 (power curve), stores the corresponding curve number in R_{00}, and stores a, b, and r^2 in R_{01}, R_{02}, and R_{03}.

THE TIME VALUE OF MONEY

The TVM program solves different problems involving time, money, and interest—the compound-interest functions. The following variables can be inputs or results.

N	The number of compounding periods or payments. (For a 30-year loan with monthly payments, $N = 12 \times 30 = 360$.)
I	The periodic interest rate as a percent. (For other than annual compounding, this represents the annual percentage rate divided by the number of compounding periods per year. For instance, 9% annually compounded monthly equals $9 \div 12 = 0.75\%$.)
PV	The present value. (This can also be an initial cash flow or a discounted value of a series of future cash flows.) Always occurs at the beginning of the first period.
PMT	The periodic payment.
FV	The future value. (This can also be a final cash flow or a compounded value of a series of cash flows.) Always occurs at the end of the *N*th period.

You can specify the timing of the payments to be either at the *end* of the compounding period (End mode) or at the *beginning* of the period (Begin mode). Begin mode sets flag 00. Ending payments are common in mortgages and direct-reduction loans; beginning payments are common in leasing.

Equation

$$0 = PV + (1 + ip)\, PMT \left[\frac{1 - (1 + i)^{-N}}{i}\right] + FV\,(1 + i)^{-N}$$

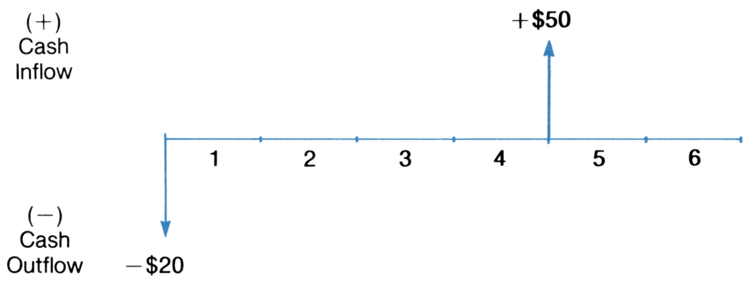
where i is the periodic interest rate as a *fraction* ($i = I/100$),
$p = 1$ in Begin mode or 0 in End mode.

Valid Input Values for Data

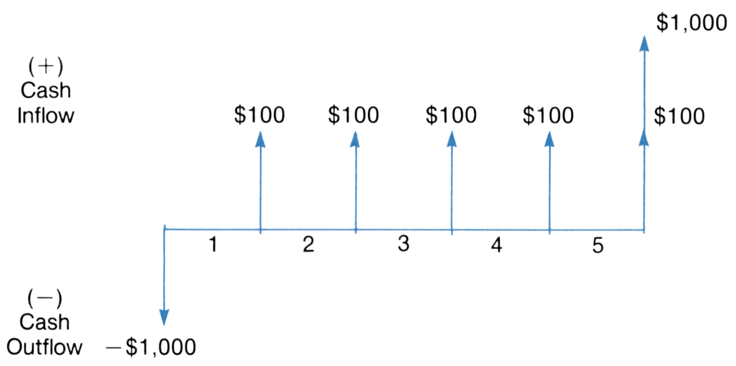
Use a *cash-flow diagram* to determine what your cash-flow inputs are and whether to specify them as *positive* or *negative*.

The cash-flow diagram is just a time-line divided into time periods. Cash flows (transactions) are indicated by vertical arrows: an *upward* arrow is *positive* for cash *received*, while a *downward* arrow is *negative* for cash *paid out*.

For example, this six-period time line shows a \$20 cash outflow initially and a \$50 cash inflow at the end of the fourth period. (Begin mode cannot be used in calculating *PV* or *FV*.)



This five-period time line shows a \$1,000 cash outflow initially and a \$100 inflow at the end of each period, ending with an additional \$1,000 inflow at the end of the fifth period.

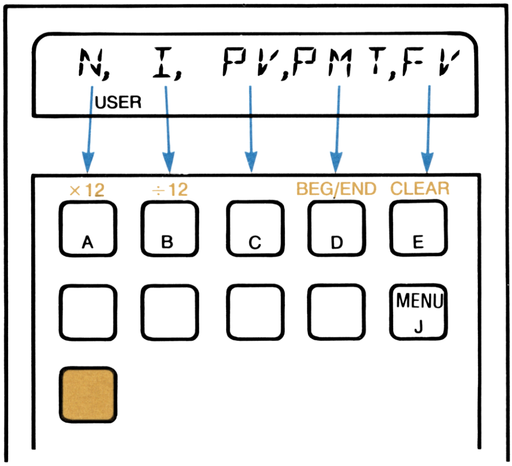


Instructions

- The program TVM will solve for any one of the variables *N*, *I*, *PV*, *PMT*, or *FV* given the other three or four, which must include either *N* or *I*. The order of entry is unimportant.*
- You should clear the financial data ([E]) before beginning a completely new calculation; otherwise, previous data that is not overwritten will be used (i.e., for the fourth, unused variable). Running the program anew also clears the financial data.
- Remember to specify cash inflows (arrow up) as positive values and cash outflows (arrow down) as negative values. The results are also given as positive or negative, indicating inflow or outflow.
- Check that the payment mode is what you want. If you see the flag 00 annunciator (a small **0** below the main display line), then Begin mode is set. If not, End mode is set. To change the mode, press [D] (a toggle). The display will then show what you have just set: **BEGIN MODE** or **END MODE.** The default is End mode (flag 00 clear).
- Remember that the interest rate must be consistent with the number of compounding periods. (An *annual* percentage rate is appropriate only if the number of compounding periods also equals the number of years.)
- You might want to set the display format for two decimal places ([FIX] 2).

---

* If you use only four variables, then the fifth must equal zero. All variables are set to zero when you first run TVM or clear the financial data ([E]), so you do not have to enter a zero in these cases.



This menu will show you which key corresponds to which function in TVM. Press [J] to recall this menu to the display at any time. This will not disturb the program in any way.

To clear the menu at any time, press [←]. This shows you the contents of the X-register, but does not end the program. You can perform calculations, then recall the main menu by pressing [J]. (However, you do not *need* to clear the program's display or recall the menu before performing calculations.)

Instruction Table for TVM

		Size: 010
Instructions	**Key In:**	**Display**
1. Start the TVM program. The menu items in the display indicate the locations of keys in the top row for *N*, *I*, *PV*, *PMT*, and *FV*.	[XEQ] [TVM] *	**N, I, PV,PMT,FV**
2. Check payment mode by looking for the **0** annunciator. (**0** means Begin mode; no **0** means End mode.) Change the mode if necessary.	■ [D] (toggles between modes) [R/S] or [J]	**END MODE** or **BEGIN MODE** **N, I, PV,PMT,FV**
3a. Input the number of compounding periods, *N*, unless *N* is what you need to find. Step 3b is a shortcut if you need to figure the number of months from a given number of years.	*N* [A] **(N)**	**N=** *N*†

Instruction Table for TVM (Continued)

Instructions	Key In:	Display
3b. Alternative to 3a: If you're working with monthly payments or monthly compounding periods for a known number of years, this step automatically figures and inputs *N* (as 12 × years). Input the number of years, *n*.	*n* ■ [A]	**N**= *12* × *n*
4a. Input the periodic interest rate, *I*, unless *I* is what you need to find. Step 4b is a shortcut if you need to figure a monthly interest rate from a given annual interest rate.	*I* [B] (**I**)	**I**=*I*
4b. Alternative to 4a: If you're working with monthly compounding periods and a known annual interest rate, this step automatically figures and inputs *I* (as annual percentage rate ÷ 12). Input the annual percentage rate, *APR*.	*APR* ■ [B]	**I**=*APR* ÷ *12*
5. Input the present value, *PV*, *unless PV* is what you need to find *or* is not a relevant variable.	*PV* [C] (**PV**)	**PV**=*input*
6. Input the amount of payment, *PMT*, *unless PMT* is what you need to find *or* is not a relevant variable.	*PMT* [D] (**PMT**)	**PMT**=*input*
7. Input the future value, *FV*, *unless FV* is what you need to find *or* is not a relevant variable.	*FV* [E] (**FV**)	**FV**=*input*
8. Now find the remaining variable by pressing its key.	[A] or [B] or [C] or [D] or [E]	**N**= *result* or **I**=*result* or **PV**=*result* or **PMT**=*result* or **FV**=*result*
9. To review (recall) any variable's value at any time:	[RCL] [A] through [E]	*value*
10. To restore the main menu (**N, I, PV,PMT,FV**) at any time (without affecting your inputs and calculations):	[J]	**N, I, PV,PMT,FV**
11. Clear old financial data before starting a new problem.	■ [E]	**N, I, PV,PMT,FV**

* To execute a program, press [XEQ] [ALPHA] *Alpha name* [ALPHA] or use a User-defined key.

† If you have a printer attached, the display automatically returns to the main menu (**N, I, PV,PMT,FV**) after printing the most recent input.

Remarks

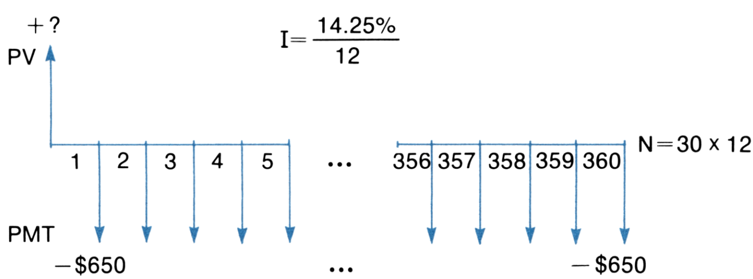
This program uses local Alpha labels (as explained in the owner's manual for the HP-41) assigned to keys [A]–[E], their shifted counterparts (except ■ [C]), and [J]. These local assignments are *overridden* by any User-key assignments you might have made to these same keys, thereby defeating this program. *Therefore be sure to clear any existing User-key assignments of these keys before using this program,* and avoid redefining these keys in the future.

The financial variable keys will only store a value if you enter it from the keyboard. If, for example, you recall a value from a register then press a variable key, the program will calculate that variable instead of storing the recalled value. To store a value that was placed in the X-register by some other means than actually keying it in, press [STO] before pressing the variable key.

Examples

A borrower can afford a $650.00 monthly payment on a 30-year, 14 ¼% mortgage. How much can he borrow? The first payment is made one month after the money is loaned. (This requires End mode.)

Cash Flows, Example 1



Keystrokes	Display	
[FIX] 2		Sets the display format used here.
[XEQ] [SIZE] 010		Optional—sets the number of storage registers needed for the program. This is not necessary if your allocation is already SIZE ⩾ 010.
[XEQ] [TVM]	**N, I, PV,PMT,FV**	Starts program. This also clears old financial data. End mode is automatically set.
30 ■ [A]	**N=360.00**	Total number of periods.
[R/S]	**N, I, PV,PMT,FV**	Restores menu (optional).
14.25 ■ [B]	**I=1.19**	Monthly interest rate.
[R/S]	**N, I, PV,PMT,FV**	Restores menu (optional).
650 [CHS] [D] **(PMT)**	**PMT=−650.00**	Monthly payment.
[R/S]	**N, I, PV,PMT,FV**	Restores menu (optional).
[C] **(PV)**	**PV=53955.92**	Maximum loan amount.

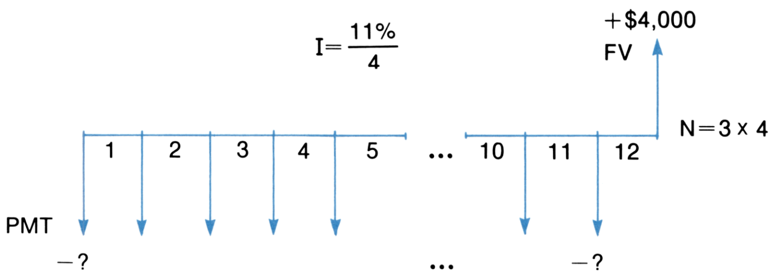
If the required mortgage is only $53,500, what is the monthly payment? (Change the *PV*, leave all other variables as they are, and solve for *PMT*.)

Keystrokes	Display	
[R/S]	**N, I, PV,PMT,FV**	Restores menu (optional).
53500 [C] **(PV)**	**PV=53500.00**	Given loan amount.
[R/S]	**N, I, PV,PMT,FV**	Restores menu (optional).
[D] **(PMT)**	**PMT=−644.51**	Monthly payment.

Notice that when you press a key [A]–[E] after keying in a value, the calculator *stores* that value in the indicated variable. However, when you press [A]–[E] *without* first keying in a value, the calculator *computes* a value for the indicated variable.

How much money must be set aside in a savings account each quarter in order to accumulate $4,000 in 3 years? The account earns 11% interest, compounded quarterly, and deposits begin immediately.

Cash Flows, Example 2



Keystrokes	**Display**	
■ [E]	N, I, PV,PMT,FV	Clears financial data. (This assumes that you are still in the TVM program.)
■ [D]	BEGIN MODE	Sets Begin mode. (The **0** annunciator should appear.)
[R/S]	N, I, PV,PMT,FV	
3 [ENTER↑] 4 [×]	12.00	Total number of
[A] (N)	N=12.00	deposits.
11 [ENTER↑] 4 [÷]	2.75	Quarterly interest
[B] (I)	I=2.75	rate.
4000 [E] (FV)	FV=4000.00	Goal.
[D] (PMT)	PMT=−278.22	Monthly deposit required.

Programming Information

The following subroutines in TVM can be used in your own programs. They find the number of periods, interest, present value, payment, or future value when given the other four parameters.

Minimum Size to Run: SIZE 010.

Flags Used: 00, 25.

Subroutine Name	Initial Registers	Final Registers	Flags to Initialize
N (*number of periods*)	$R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	X-register $= N$ $R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	SF 00 for Begin mode CF 00 for End mode
*I (*interest*)	$R_{01} = N$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	$X = I$ $R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	SF 00 for Begin mode CF 00 for End mode
PV (*present value*)	$R_{01} = N$ $R_{02} = I$ $R_{04} = PMT$ $R_{05} = FV$	$X = PV$ $R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	SF 00 for Begin mode CF 00 for End mode
PMT (*payment value*)	$R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{05} = FV$	$X = PMT$ $R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	SF 00 for Begin mode CF 00 for End mode
FV (*future value*)	$R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$	$X = FV$ $R_{01} = N$ $R_{02} = I$ $R_{03} = PV$ $R_{04} = PMT$ $R_{05} = FV$	SF 00 for Begin mode CF 00 for End mode

Comments. To use these subroutines, load the four appropriate registers, set (or clear) flag 00 for Begin (or End) mode, then execute the desired subroutine. It returns the desired value to the X-register and stores it in the corresponding register.

PROGRAM INDEX

HEWLETT
PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

Reorder Number
00041-90546

Printed in U.S.A. 8/85