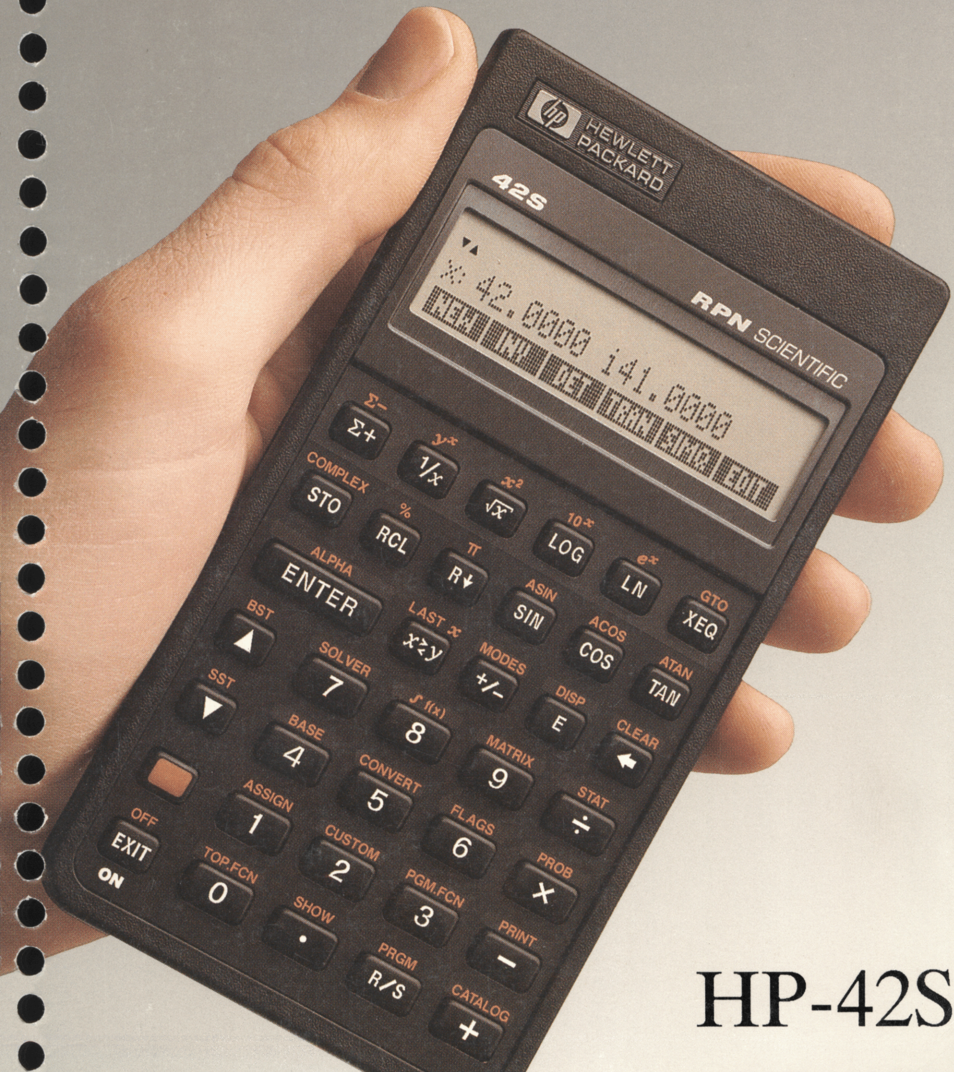
HEWLETT-PACKARD

RPN Scientific

Programming Examples and Techniques



HP-42S

HEWLETT PACKARD

Help Us Help You!

Please take a moment to complete this postage-paid card, tear it out and put it in the mail. Your responses and comments will help us better understand your needs and will provide you with the best procedures to solve your problems. Thank you!

HELP US HELP YOU!

Book: **Programming Examples and Techniques** Date acquired: ____________

Name ____

Street ____

City, State, Zip ____

Phone (________) ____________________ Business ______ or Home ______

1. What calculator will you use this book with?
   009 ☐ **HP-42S** 006 ☐ **Other** ____________________
2. How many other HP solution books have you bought for this calculator? ______
3. What is your OCCUPATION?
   101 ☐ **Student** 103 ☐ **Professional** 109 ☐ **Other** ____________
4. Where did you purchase this book?
   403 ☐ **Bookstore** 404 ☐ **Discount or Catalog Store**
   407 ☐ **Mail Order** 410 ☐ **HP Direct** 411 ☐ **Other** ____________
5. How did you first hear about this book?
   501 ☐ **HP Owner** 503 ☐ **Advertising** 506 ☐ **Salesperson** 507 ☐ **Brochure**
   508 ☐ **Other** ____________________
6. To what degree did this book influence your calculator purchase decision?
   601 ☐ **Major Influence** 602 ☐ **Minor Influence** 603 ☐ **No Influence**
7. How well does this book cover the material you expected?
   701 ☐ **Good** 702 ☐ **Moderate** 703 ☐ **Low**
8. What level of knowledge is required to make use of the topics in this book?
   801 ☐ **High** 802 ☐ **Medium** 803 ☐ **Low**
9. How clearly was the material in this book presented?
   901 ☐ **Good** 902 ☐ **Moderate** 903 ☐ **Low**
10. How would you rate the value of this book for your money?
    111 ☐ **High** 112 ☐ **Medium** 113 ☐ **Low**

Comments: (Please comment on improvements and additional applications or subjects you would like HP to cover in this or another solution book.) ____________

____

____

____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL   PERMIT NO. 40   CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
CALCULATOR MARKET RESEARCH
1000 NE CIRCLE BLVD.
CORVALLIS OR 97330-9988

HP-42S
RPN Scientific

Programming Examples and Techniques

HEWLETT PACKARD

Edition 1   July 1988
Reorder Number   00042-90020

Notice

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

Printing History

Edition 1	July 1988	Mfg. No. 00042-90019

Contents

List of Examples

The following list groups the examples by chapter.

7 Graphics and Plotting

How to Use This Manual

Welcome to the *Programming Examples and Techniques* manual for your HP-42S calculator. This manual builds on concepts introduced to you in the *HP-42S Owner's Manual* so that you can more fully utilize your calculator's powerful problem-solving capabilities. This manual focuses on the following subjects:

- Programming techniques for the HP-42S.
- Enhancing existing HP-41 programs.
- Using the HP-42S built-in applications:
    - The Solver.
    - Integration.
    - Matrices.
    - Statistics.
- Building and printing graphics patterns and plots.

There are many examples in this manual. We feel that the best way to help you gain expertise with your calculator is to show you how to solve practical problems in mathematics, science, engineering, and finance. Many of these problems are solved using programs. Chapter 1, "Programming," addresses the task of creating programs with the HP-42S. It further develops material presented to you in chapters 8 through 10 of the owner's manual.

Chapter 2 specifically addresses the topic of enhancing programs written for the HP-41 calculator. It builds on the material introduced in chapter 11 of your owner's manual.

Chapters 3 through 6 further develop the built-in applications discussed in chapters 12 through 15 of the owner's manual. If you wish to learn more about matrix operations, for example, you can turn directly to chapter 5, "Matrices," without working through the preceding chapters. However, since many of the examples in the manual are programmed solutions to problems, you should first review chapter 1.

Chapter 7 describes how to generate graphics patterns and plots using the HP-42S calculator and, in several examples, the optional HP 82240A Infrared Printer. It builds on the material presented in chapter 7 of the owner's manual.

The notations in this manual are consistent with those in the owner's manual:

- Plain typeface is used for numbers and Alpha characters in keystroke sequences: 1.2345, ABCD.
- Black keyboxes are used for primary keyboard functions in keystroke sequences: [EXIT].
- Orange keyboxes preceded by the orange shift key are used for secondary (shifted) functions in keystroke sequences: ■[ASSIGN].
- Menu keyboxes are used for functions executed from a menu in keystroke sequences: CLP.
- Capital letters are used for any function that is referenced in text: CLP.
- Capital letters are used for program names that are referenced in text: SSS.
- Italic letters are used for variable names that are referenced in text: *STEP*
- Dot matrix typeface is used for program listings: `01 LBL "AREA"`.

At the beginning of each example, it is assumed that the stack registers (X-, Y-, Z-, and T-registers) are clear (contain the value 0). It is also assumed that the value of each variable in the examples is 0. *Your display may sometimes differ from the displays in the manual.* However, if you execute the keystroke sequences as they are shown in the examples, the values of the stack registers and variables in your calculator at the start of the examples will not affect the answers you obtain.

Some examples include optional instructions to print results with the HP 82240A Infrared Printer. If you have a printer and execute these instructions, you will not see some of the subsequent displays in the example. These displays *will* be printed.

1

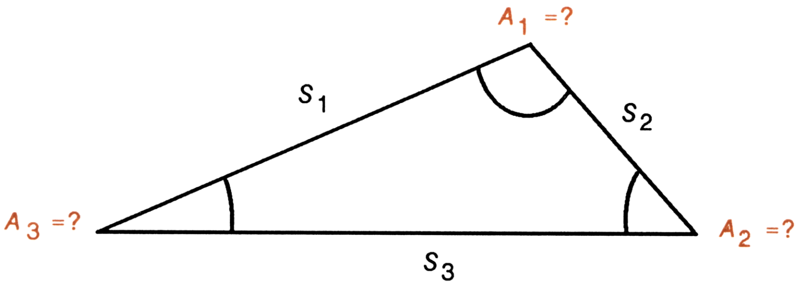
Programming

Your calculator is a powerful and easy-to-use tool for creating and executing programs. This chapter builds on programming methods introduced to you in chapters 8 through 10 of your owner's manual. Specifically, this chapter addresses:

- Simple programming.
- Branching.
- Looping controlled by a counter.
- Indirect addressing.
- Flags in programs.
- Error trapping.

Simple Programming

The program SSS in this section finds the values of the three angles of a triangle when the values of the three sides are known. (The annotated listing is on pages 17 through 18.)



When the dimensions of the three sides (S_1, S_2, and S_3) of a triangle are known, the following equations are used to calculate the three angles (A_1, A_2, and A_3).

$$A_3 = 2 \text{ arc cos} \left[\frac{\sqrt{P(P - S_2)}}{(S_1 S_3)} \right] \text{ where } P = \frac{(S_1 + S_2 + S_3)}{2}$$

$$A_2 = 2 \text{ arc cos} \left[\frac{\sqrt{P(P - S_1)}}{(S_2 S_3)} \right]$$

$$A_1 = \text{arc cos} [-\cos (A_3 + A_2)]^*$$

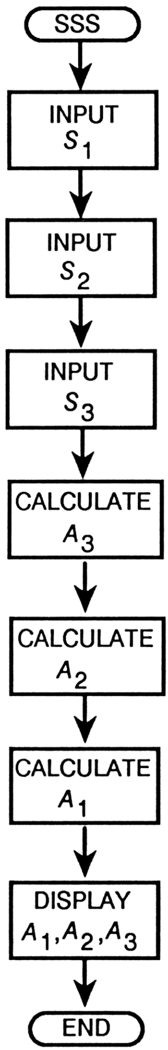
These equations form the main body of SSS.

Flowcharting

A flowchart is a graphical outline of a program. Flowcharts are used in this manual to help you understand how programs solve problems. Flowcharts can also help you design your own programs by breaking them down into smaller groups of instructions. The flowchart can be as simple or as detailed as you like. Flowcharts are drawn linearly, from top to bottom, representing the general flow of the program from beginning to end.

* This expression for A_1 enables you to calculate A_1 in any angular mode.

Here is a flowchart for one possible program solution for the side-side-side triangle problem.



This manual uses the following conventions for flowchart symbols:

- An oval represents the *beginning* or *end* of a routine. This can be the beginning or end of a program, a subroutine, or a counter-controlled loop within a program.
- A circle represents a *program label.* It also represents a *GTO instruction* to a program label from another point in the program. (This convention reduces the need for connecting lines that can make the flowchart difficult to read.)
- A rectangle represents a *functional operation* in the program.
- A diamond represents a *decision* the program makes based on a comparison of two values (or based on the status of a flag).
- A triangle represents a decision the *user* (that's you) makes by selecting one of several possible program routines, each of which performs a different task.

Defining the Program

Program SSS begins with a *global label* and ends with an END instruction. These two instructions define the beginning and end of the program.

```
01 LBL "SSS"
     :
45 END
```

Prompting for Data Input

SSS *prompts* you for data input (prompts you for the three known values of the sides of the triangle).

```
02 INPUT "S1"
03 INPUT "S2"
04 INPUT "S3"
```

Displaying Program Results

SSS concludes by *displaying* (or printing) the calculated results (the three angles).

```
41 SF 21
42 VIEW "A1"
43 VIEW "A2"
44 VIEW "A3"
```

This section of the program begins by setting flag 21, the Printer Enable flag. When flag 21 is set, a VIEW (or AVIEW) instruction is:

- *Printed and displayed* if you have executed PRON. Program execution does not halt when a message is displayed; a subsequent VIEW (or AVIEW) instruction *erases* the current message. When you set flag 21 and execute PRON, and then execute a program that has a sequence of VIEW (or AVIEW) instructions, you must have a printer present and turned on to record each message; you'll see *only* the last message in the display.
- *Displayed* by the calculator if you have executed PROFF. (PROFF is the default mode for the calculator. You need to execute PROFF only if you have previously executed PRON.) When you set flag 21 in PROFF mode, program execution halts after each VIEW (or AVIEW) instruction and must be resumed by pressing [R/S].

Helpful hints for keying in programs:

1. If the variables you are using in your program do not already exist, create them *before* you select Program-entry mode (by pressing 0 [STO] *variable* for each variable). When you subsequently key in a STO, RCL, INPUT, or VIEW instruction during program entry and are prompted for a register or variable, the existing variables (including the ones you just created) are displayed in the variable-catalog menu. You only need to press the corresponding menu key, rather than *type* the variable name.
2. In Program-entry mode, first key in all the global label instructions in your program (by pressing ■ [PGM.FCN] LBL *label* for each label). When you subsequently key in branch instructions and are prompted for a label, the existing global labels (including the ones you just created) are displayed in the program-catalog menu. You only need to press the corresponding menu key, rather than type the name.

Longer programs in this manual are preceded by instructions that list the variables and labels to create for program entry.

To key in SSS: Create variables *S1*, *S2*, *S3*, *A1*, *A2*, *A3*, and *P* before program entry.

Here is an annotated listing of SSS.

Program:	**Comments:**
`00 { 115-Byte Prgm }` `01 LBL "SSS"`	Line 01: Define the beginning of the program.

`02 INPUT "S1"`
`03 INPUT "S2"`
`04 INPUT "S3"`
	Lines 02–04: Prompt for the values of the three sides and store the values in named variables.

`05 RCL "S1"`
`06 RCL+ "S2"`
`07 RCL+ "S3"`
`08 2`
`09 ÷`
	Lines 05–40: Calculate A_1, A_2, and A_3. Store the values in named variables.

```
10 STO "P"
11 X↑2
12 LASTX
13 RCL× "S2"
14 -
15 RCL "S1"
16 RCL× "S3"
17 ÷
18 SQRT
19 ACOS
20 2
21 ×
22 STO "A3"
23 SIN
24 RCL "P"
25 X↑2
26 LASTX
27 RCL× "S1"
28 -
29 RCL÷ "S2"
30 RCL÷ "S3"
31 SQRT
32 ACOS
33 2
34 ×
35 STO "A2"
36 RCL+ "A3"
37 COS
38 +/-
39 ACOS
40 STO "A1"
```

```
41 SF 21
42 VIEW "A1"
43 VIEW "A2"
44 VIEW "A3"
```

Lines 41 – 44: Display (or print) the calculated results.

```
45 END
```

Line 45: End the program.

Executing the Program

You can execute SSS by using any one of the following keystroke sequences.

Using the Program Catalog. The global label SSS was automatically placed in the program catalog when you keyed in program line 01. You can execute the program by pressing

■ [CATALOG] PGM SSS

This sequence requires a minimum of four keystrokes, depending on where label SSS is in the program catalog. (If you have created more than five programs subsequent to SSS, use the [▼] key to find label SSS.)

Using XEQ. When you press [XEQ], the program-catalog menu is automatically displayed. Thus, you can execute SSS by pressing

[XEQ] SSS

This sequence requires a minimum of two keystrokes, depending on where label SSS is in the program catalog.

Using the CUSTOM Menu. Alternately, you can assign SSS to the CUSTOM menu by pressing

■ [ASSIGN] PGM SSS

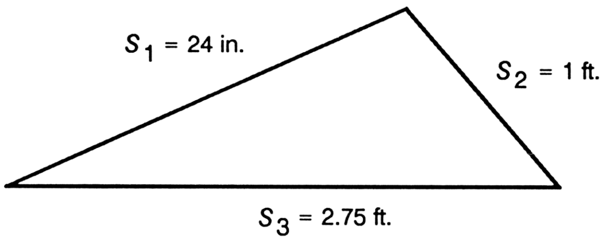
and then the desired menu key.

The program can now be executed directly from the CUSTOM menu by pressing

■ [CUSTOM] SSS

This sequence requires three keystrokes when you first select the CUSTOM menu, and only one keystroke on subsequent executions if you stay in the current row of the menu.

Example: Executing a Program from the CUSTOM Menu. Find the angles (in degrees) of the following triangle.



Assign SSS to the CUSTOM menu. Set the angular mode to Degrees. Execute PRON if you have a printer and want to print the results. Begin program execution.

■ [ASSIGN] PGM SSS ■ [MODES] DEG (■ [PRINT] [▲] PON) ■ [CUSTOM] SSS	S1?0.0000 SSS

Enter the value for S_1 (in feet) and continue program execution.

24 [ENTER] 12 [÷] [R/S]	S2?0.0000 SSS

Enter the value for S_2, then for S_3. The program now calculates the three angles and displays *A1*, the first result. (If you have executed PRON to print the results, you won't see the next two displays.)

1 [R/S] 2.75 [R/S]

```
A1=129.8384
SSS
```

Continue program execution to see A_2.

[R/S]

```
A2=33.9479
SSS
```

Continue program execution to see A_3.

[R/S]

```
A3=16.2136
SSS
```

Exit from the program.

[EXIT]

```
Y: 0.2792
X: 129.8384
```

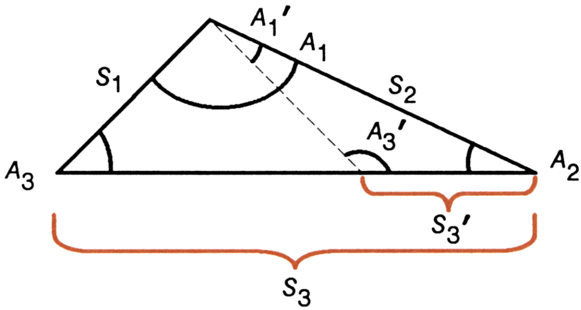
Branching

A branch instruction enables program execution to jump to a different location in program memory. A branch can be:

- Conditional (based on a test).
- Unconditional (used typically to call a subroutine that, on completion, returns program execution to the main program).
- Menu-controlled (executed by you from a programmable menu).

Conditional Branching

The program SSA on pages 24 through 25 in this section illustrates the use of conditional branching. SSA finds the two unknown angles and the unknown side of a triangle when two sides and the adjacent angle (S_1, S_2, and A_2) are known.



The equations used to calculate A_3, A_1, and S_3 are

$$A_3 = \text{arc sin}\left[\left(\frac{S_2}{S_1}\right)\sin A_2\right]$$

$$A_1 = \text{arc cos}\left[-\cos\left(A_2 + A_3\right)\right]$$

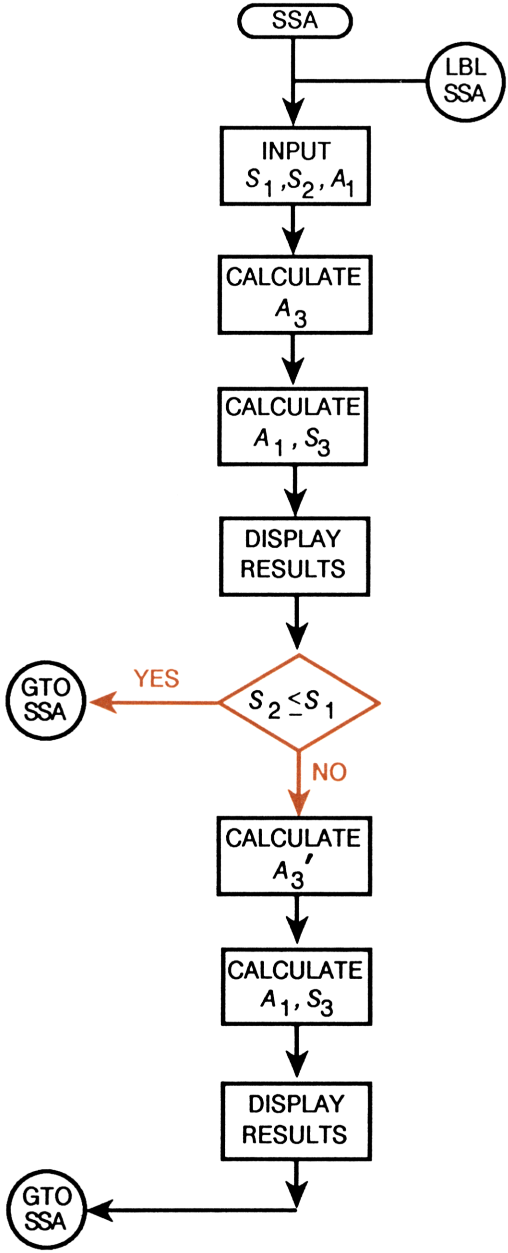
$$S_3 = S_1 \cos A_3 + S_2 \cos A_2$$

Note from the drawing that two possible solutions exist if S_2 is greater than S_1 and A_3 does not equal 90°. This leads to a fourth equation.

$$A_3' = \text{arc cos}\left(-\cos A_3\right)$$

SSA calculates both possible answer sets.

Here is a flowchart for the program.


SSA
LBL SSA
INPUT S_1, S_2, A_1
CALCULATE A_3
CALCULATE A_1, S_3
DISPLAY RESULTS
$S_2 \le S_1$
YES
GTO SSA
NO
CALCULATE A_3'
CALCULATE A_1, S_3
DISPLAY RESULTS
GTO SSA

Observe from the flowchart that the program calculates the first answer set, then compares the values of S_1 and S_2. Depending on the result of the comparison, the program either returns to label SSA or calculates the second answer set. SSA accomplishes this with a *conditional branch*. The corresponding keystrokes are highlighted in the following annotated listing. (This conditional branch is based on a *number* test. Later in this chapter, you'll write programs that make conditional branches based on *flag* tests.)

To key in SSA: Create variables *S1*, *S2*, *S3*, *A1*, *A2*, and *A3* before program entry. (These variables already exist if you keyed in program SSS.)

Program: **Comments:**

```
00 { 157-Byte Prgm }
01 LBL "SSA"

02 SF 21
```

```
03 INPUT "S1"
04 INPUT "S2"
05 INPUT "A2"
```

Lines 03–05: Input the known variables.

```
06 SIN
07 RCL× "S2"
08 RCL÷ "S1"
09 ASIN
10 STO "A3"
11 RCL+ "A2"
12 COS
13 +/-
14 ACOS
15 STO "A1"
16 RCL "A2"
17 COS
18 RCL× "S2"
19 RCL "A3"
20 COS
21 RCL× "S1"
22 +
```

Lines 06–23: Calculate the unknown variables.

```
23 STO "S3"
```

```
24 VIEW "A1"
25 VIEW "S3"
26 VIEW "A3"
```

Lines 24–26: Display (or print) the unknown variables.

```
27 RCL "S1"
28 RCL "S2"
29 X≤Y?
30 GTO "SSA"
```

Lines 27–30: Test if S_2 is less than or equal to S_1. If so, return to the beginning of the program. If not, calculate the second answer set.

```
31 RCL "A3"
32 COS
33 +/-
34 ACOS
35 STO "A3"
36 RCL+ "A2"
37 COS
38 +/-
39 ACOS
40 STO "A1"
41 RCL "A2"
42 COS
43 RCL× "S2"
44 RCL "A3"
45 COS
46 RCL× "S1"
47 +
48 STO "S3"
```

Lines 31–48: Calculate the second answer set.

```
49 VIEW "A1"
50 VIEW "S3"
51 VIEW "A3"
52 GTO "SSA"
```

Lines 49–52: Display the second answer set and return to the beginning of the program.

```
53 END
```

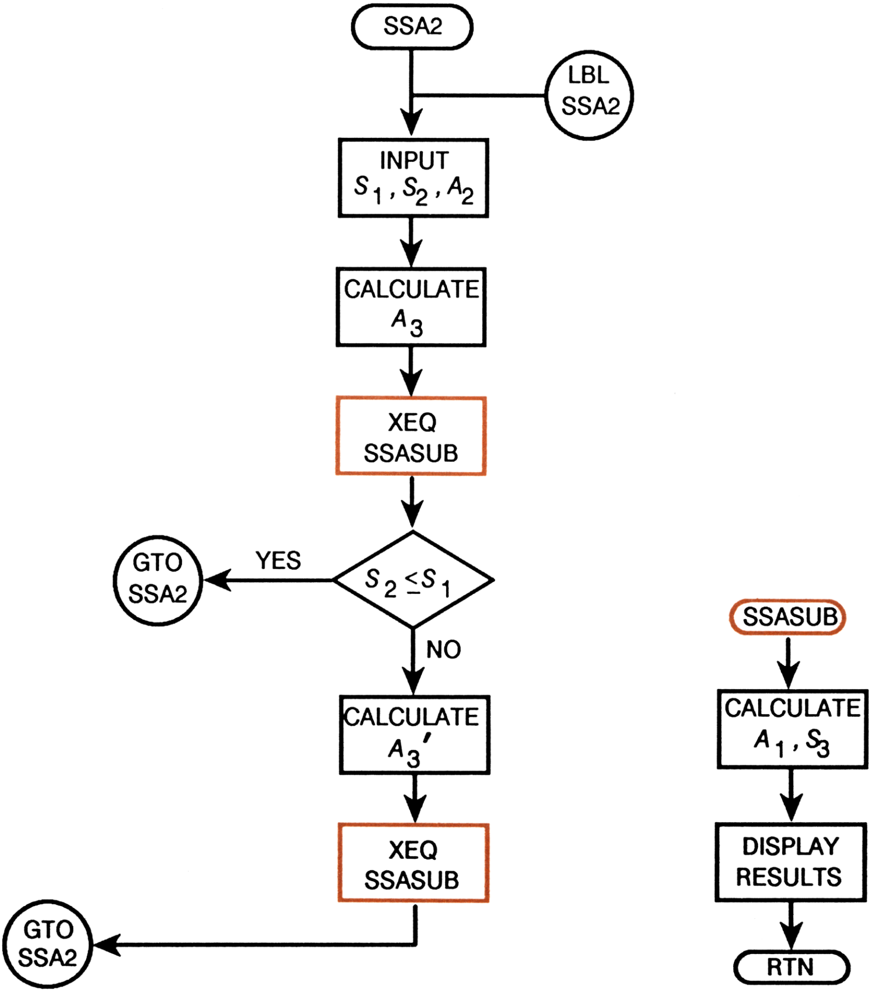
Subroutines

A routine is a set of program steps defined by a local or global label and a RTN or END instruction. (Programs SSS and SSA are routines.) A routine becomes a *subroutine* when it is *called by* (executed from) another routine using an XEQ instruction. After the subroutine has been executed, the RTN or END instruction at the end of the subroutine *returns* program execution to the main routine.

Notice that SSA calculates the second answer set (if there is one) by first calculating A_3'. It then calculates the remaining unknowns using the *same* equations that were used to calculate the first answer set and displays the second answer set using the same instructions that were used to display the first answer set. By placing these shared instructions in a subroutine, the program becomes:

- Shorter.
- Easier to read.
- Easier to write.
- Easier to edit.

Here is a flowchart for a new program SSA2 that uses a subroutine.



The corresponding program lines are highlighted in the following annotated listing.

To key in SSA2:

1. Create variables *S1*, *S2*, *S3*, *A1*, *A2*, and *A3* before program entry.
2. Create label SSASUB when you begin program entry.

Program: **Comments:**

```
00 { 137-Byte Prgm }
01 LBL "SSA2"
02 SF 21
03 INPUT "S1"
04 INPUT "S2"
05 INPUT "A2"
```

```
06 SIN
07 RCL× "S2"
08 RCL÷ "S1"
09 ASIN
```

Lines 06–09: Calculate A_3.

```
10 XEQ "SSASUB"
```

Line 10: Call subroutine SSASUB to calculate A_1 and S_3. This unconditional branch uses an XEQ instruction; the next encountered RTN (or END) instruction will transfer program execution back to line 11. (Now follow the branch to line 21.)

```
11 RCL "S1"
12 RCL "S2"
13 X≤Y?
14 GTO "SSA2"
```

Lines 11–14: If S_2 is less than or equal to S_1, return to the beginning of the program. If not, calculate the second answer set.

```
15 RCL "A3"
16 COS
17 +/-
18 ACOS
```

Lines 15–18: Calculate A_3'.

```
19 XEQ "SSASUB"
20 GTO "SSA2"
```

Lines 19–20: Call subroutine SSASUB to calculate A_1' and S_3'. Then return to the beginning of the program.

```
21 LBL "SSASUB"
22 STO "A3"
23 RCL+ "A2"
24 COS
```

Subroutine SSASUB, lines 21–39: Calculate the values A_1 and S_3 (A_1' and S_3' in the second answer set), and display the results.

```
25 +/-
26 ACOS
27 STO "A1"
28 RCL "A2"
29 COS
30 RCL× "S2"
31 RCL "A3"
32 COS
33 RCL× "S1"
34 +
35 STO "S3"
36 VIEW "A1"
37 VIEW "S3"
38 VIEW "A3"
39 RTN

40 END
```

SSA2 is 13 lines shorter than SSA and 20 bytes shorter than SSA.

Nested Subroutines. The program TRI∡ in the following section organizes each of the five possible triangle solutions in subroutines labeled A through E. Refer to the flowchart for TRI∡ on pages 30–31, and note that subroutine B, which calculates the solution to the SSA initial condition, itself calls subroutine SSASUB to calculate A_2 and S_3. In TRI∡, subroutine SSASUB is *nested* in subroutine B. When subroutine SSASUB is called by subroutine B, there are two *pending* subroutines. The HP-42S can have up to eight pending subroutines.

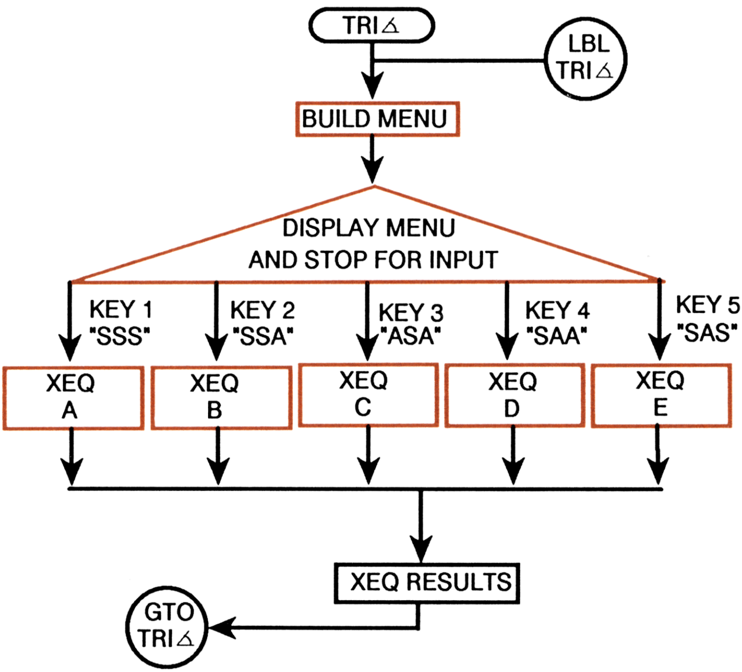
Menu-Controlled Branching

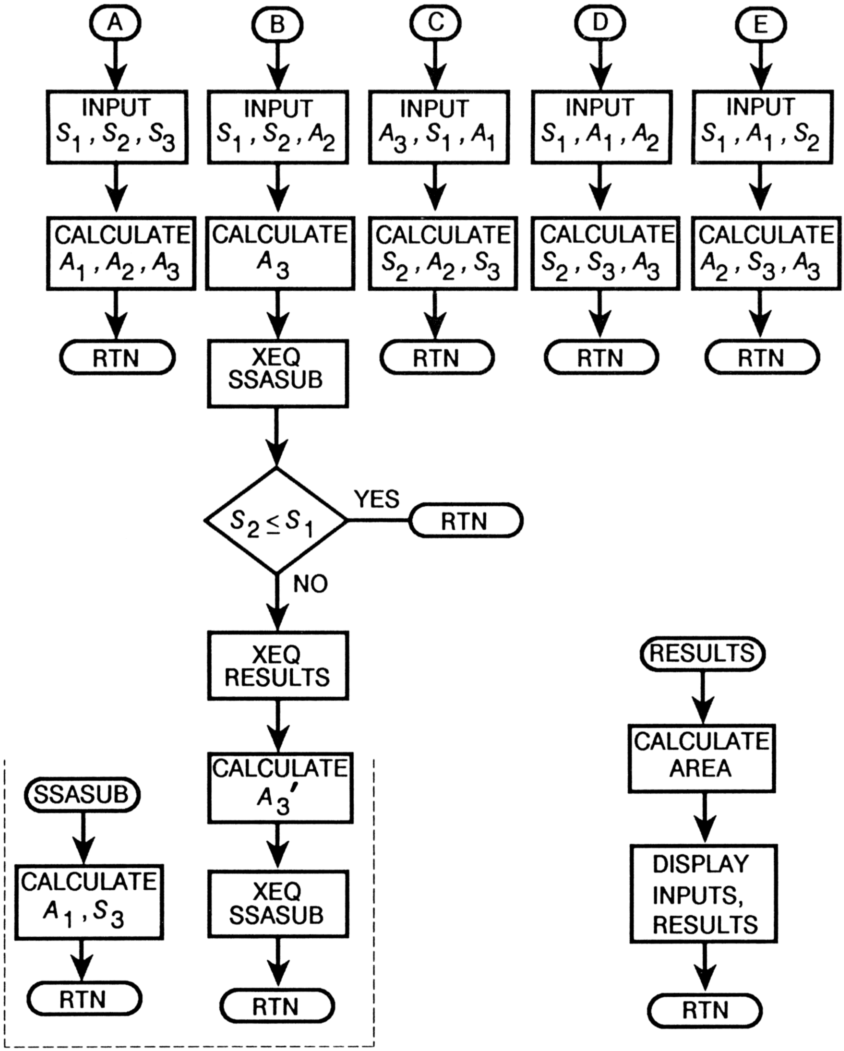
Programmable menus enable *you* to make a decision during a program, prompted by labeled menu keys that cause branches to new locations in program memory. Using KEY XEQ or KEY GTO instructions (which act just like XEQ and GTO instructions), any label in program memory can be made the target of a programmable menu key. When MENU and STOP instructions are subsequently executed, program execution is suspended, the programmable menu is displayed, and keys 1 through 9 (the six top-row keys, plus the [▲], [▼], and [EXIT] keys) assume their menu definitions.

The previous two programs, SSS and SSA, each calculated one of the five triangle solutions. The other solutions respectively find:

- S_2, A_2, and S_3 (when A_3, S_1, and A_1 are known).
- S_2, S_3, and A_3 (when S_1, A_1, and A_2 are known).
- A_2, S_3, and A_3 (when S_1, A_1, and S_2 are known).

Here is a flowchart for a program named TRI∡. TRI∡ organizes each of the five solutions in a subroutine, builds a programmable menu, and allows you to select any solution by pressing the corresponding menu key.




A
B
C
D
E
INPUT S_1, S_2, S_3
INPUT S_1, S_2, A_2
INPUT A_3, S_1, A_1
INPUT S_1, A_1, A_2
INPUT S_1, A_1, S_2
CALCULATE A_1, A_2, A_3
CALCULATE A_3
CALCULATE S_2, A_2, S_3
CALCULATE S_2, S_3, A_3
CALCULATE A_2, S_3, A_3
RTN
XEQ SSASUB
RTN
RTN
RTN
$S_2 \leq S_1$
YES
RTN
NO
XEQ RESULTS
CALCULATE A_3'
XEQ SSASUB
RTN
SSASUB
CALCULATE A_1, S_3
RTN
RESULTS
CALCULATE AREA
DISPLAY INPUTS, RESULTS
RTN

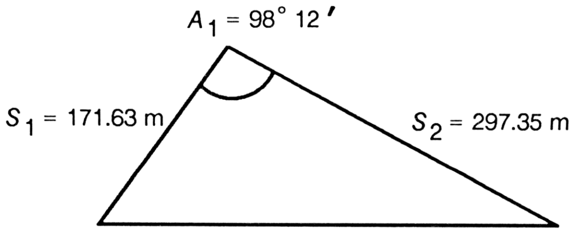
The triangle symbol in the flowchart indicates where the program stops to display the menu. You choose which solution you want to execute by pressing the corresponding menu key.

Here are the corresponding program lines.

Program:	**Comments:**
03 "SSS" 04 KEY 1 XEQ A 05 "SSA" 06 KEY 2 XEQ B 07 "ASA" 08 KEY 3 XEQ C 09 "SAA" 10 KEY 4 XEQ D 11 "SAS" 12 KEY 5 XEQ E	Lines 03 – 12: Build the menu keys. (For example, lines 03 and 04 label menu key 1 with the Alpha string SSS and define that key to execute a branch to label A.)
13 MENU 14 STOP 15 XEQ "RESULTS" 16 GTO "TRI∡"	Lines 13 – 16: Select the menu (line 13) and suspend program execution (line 14). (The menu is displayed when program execution halts.) After execution of any subroutine A through E, call subroutine RESULTS to display the results (line 15). Then return to label TRI∡ at the start of the program (line 16).

The complete listing of TRI∡ is on pages 60 – 65 at the end of this chapter.

Example: A Programmable Menu. A surveyor needs to find the area and dimensions of a triangular land parcel. From point A, he measures the distance to points B and C, and the angle between AB and AC.



This is an SAS (side-angle-side) problem.

Set the angular mode to Degrees. (Execute PRON if you want to print the results.) Begin program execution.

■ [MODES] DEG
(■ [PRINT] [▲] PON)
[XEQ] TRI∡

```
x: 0.0000
SSS SSA ASA SAA SAS
```

Select the SAS routine by pressing menu key 5.

SAS

```
S1?0.0000
SSS SSA ASA SAA SAS
```

Key in the value for S_1 and continue program execution.

171.63 [R/S]

```
A1?0.0000
SSS SSA ASA SAA SAS
```

Key in the values for A_1 (you need to convert A_1 to its decimal equivalent) and S_2. The program calculates the unknowns and displays the initial known values and calculated results.

98.12 ■ [CONVERT] →HR
[R/S] 297.35 [R/S]

```
S1=171.6300
x: 25,256.2094
```

Press [R/S] three times to see A_2.

[R/S] [R/S] [R/S]

```
A2=27.8270
x: 25,256.2094
```

Press [R/S] again to see S_3.

[R/S]

```
S3=363.9118
x: 25,256.2094
```

Press [R/S] again to see A_3.

[R/S]

```
A3=53.9730
x: 25,256.2094
```

Press [R/S] again to see *AREA*.

[R/S]

```
AREA=25,256.2094
x: 25,256.2094
```

Press [R/S] again to display the menu.

[R/S]

```
x: 25,256.2094
 SSS  SSA  ASA  SAA  SAS
```

End the program.

[EXIT]

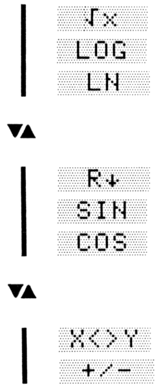
```
y: 27.8270
x: 25,256.2094
```

Multirow Menus. The preceding program, TRI∡, builds menu labels for five of the six top-row keys, and assigns a KEY XEQ instruction to each labeled key.

A multirow menu has *more than one row* of labeled keys. (For example, the CLEAR menu has two rows.) When you enter a multirow menu, the [▼] and [▲] keys enable you to move to each row in the menu. (The ▼▲ annunciator appears in the display to show you that these keys may be used to display more rows.)

You can *emulate* a multirow menu in a program by assigning KEY GTO instructions to menu key 7 (the [▲] key) and menu key 8 (the [▼] key). (KEY GTO or KEY XEQ instructions for menu keys 7 and 8 also *automatically* turn on the ▼▲ annunciator in the display.)

Consider the following simple menu of calculator functions.



Here is a program that emulates this multirow menu.

To key in ROW1:

1. Create labels ROW1, ROW2, and ROW3 when you begin program entry.
2. Note that program lines 03, 05, 07, 16, 18, 20, 29, and 31 are Alpha strings.

Program:	**Comments:**
00 { 184-Byte Prgm } 01 LBL "ROW1"	
02 CLMENU 03 "√" 04 KEY 3 XEQ 01 05 "LOG" 06 KEY 4 XEQ 02 07 "LN" 08 KEY 5 XEQ 03 09 KEY 7 GTO "ROW3" 10 KEY 8 GTO "ROW2" 11 MENU 12 STOP 13 GTO "ROW1"	Lines 01–13: Clear the current menu definitions, then build and display the first row of the menu. Assign branch instructions to keys 7 and 8 (the [▲] and [▼] keys) to the previous and succeeding rows respectively (lines 09–10).

```
14 LBL "ROW2"
15 CLMENU
16 "R↓"
17 KEY 3 XEQ 04
18 "SIN"
19 KEY 4 XEQ 05
20 "COS"
21 KEY 5 XEQ 06
22 KEY 7 GTO "ROW1"
23 KEY 8 GTO "ROW3"
24 MENU
25 STOP
26 GTO "ROW2"
```

Lines 14–26: Clear the current menu definitions, then build and display the second row of the menu. Assign branch instructions to keys 7 and 8 to the previous and succeeding rows respectively (lines 22–23).

```
27 LBL "ROW3"
28 CLMENU
29 "X<>Y"
30 KEY 3 XEQ 07
31 "+/-"
32 KEY 4 XEQ 08
33 KEY 7 GTO "ROW2"
34 KEY 8 GTO "ROW1"
35 MENU
36 STOP
37 GTO "ROW3"
```

Lines 27–37: Clear the current menu definitions, then build and display the third row of the menu. Assign branch instructions to keys 7 and 8 to the previous and succeeding rows respectively (lines 33–34).

```
38 LBL 01
39 SQRT
40 RTN
41 LBL 02
42 LOG
43 RTN
44 LBL 03
45 LN
46 RTN
47 LBL 04
48 R↓
49 RTN
```

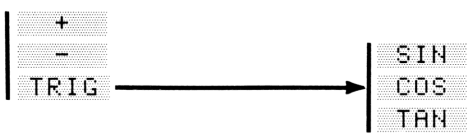
Subroutines 01–08, lines 38–61: Execute the calculator functions corresponding to each menu label.

```
50 LBL 05
51 SIN
52 RTN
53 LBL 06
54 COS
55 RTN
56 LBL 07
57 X<>Y
58 RTN
59 LBL 08
60 +/-
61 RTN

62 END
```

Nested Menus. In many menus, one or more of the six *top-row menu keys* bring up a new menu called a nested, or submenu. For example, in the PGM.FCN menu, when you press the X?0 menu key, a nested menu of related functions (X=0?, X≠0?, ..., X≥0?) is displayed. To return to the main menu, you press the [EXIT] key.

You can *emulate* a nested menu in a program by assigning a KEY GTO instruction to any labeled top-row menu key. Consider the following simple menu of calculator functions.



Here is a program that emulates this menu structure.

To key in LVL1:

1. Create labels LVL1 and LVL2 when you begin program entry.
2. Note that lines 03, 05, 07, 14, 16, and 18 are Alpha strings.

Program: **Comments:**

```
00 { 108-Byte Prgm }
01 LBL "LVL1"
```

```
02 CLMENU
03 "+"
04 KEY 2 XEQ 01
05 "-"
06 KEY 3 XEQ 02
07 "TRIG"
08 KEY 5 GTO "LVL2"
09 MENU
10 STOP
11 GTO "LVL1"
```

Lines 01 – 11: Build and display the primary level of the menu. Assign to key 3 (labeled TRIG) a branch instruction to label LVL2 to build the nested menu (line 08).

```
12 LBL "LVL2"
13 CLMENU
14 "SIN"
15 KEY 4 XEQ 11
16 "COS"
17 KEY 5 XEQ 12
18 "TAN"
19 KEY 6 XEQ 13
20 KEY 9 GTO "LVL1"
21 MENU
22 STOP
23 GTO "LVL2"
```

Lines 12 – 23: Build and display the nested menu. Assign a branch instruction to key 9 (the [EXIT] key) back to label LVL1 (line 20).

```
24 LBL 01
25 +
26 RTN
27 LBL 02
28 -
29 RTN
30 LBL 11
31 SIN
32 RTN
33 LBL 12
34 COS
```

Subroutines 01, 02, and 11 – 13, lines 24 – 38: Execute the calculator functions corresponding to each menu label.

```
35 RTN
36 LBL 13
37 TAN
38 RTN

39 END
```

Controlled Looping

A controlled loop is a loop that is executed a specified number of times. You can build a controlled loop with a local or global label, an ISG or DSE instruction, and a GTO instruction.

The program DISPL in this section uses a controlled loop to calculate successive linear displacements of an object traveling at a constant velocity.

The equation of motion for constant velocity on a smooth surface is

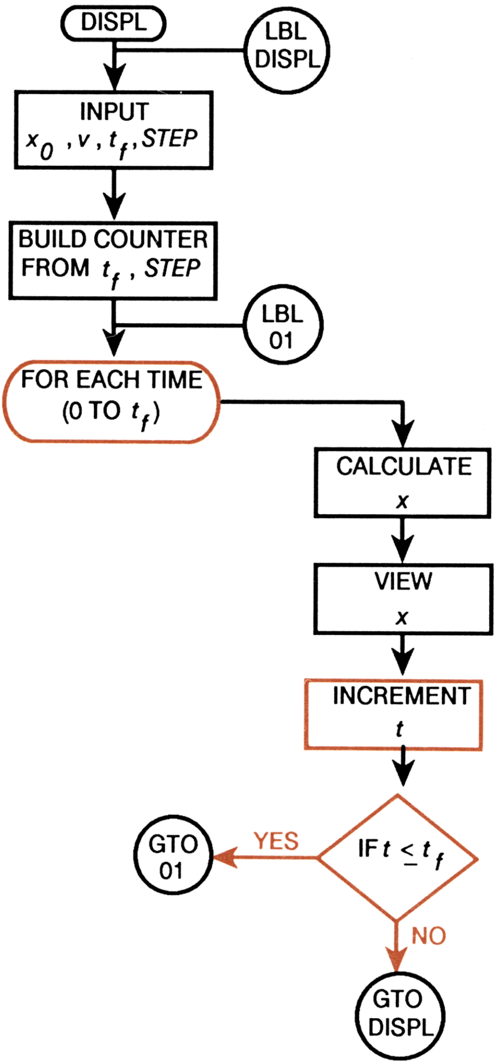
$$x = x_0 + vt$$

where:

x is the total displacement.
x_0 is the initial position.
v is the velocity.
t is the elapsed time.

DISPL calculates the displacement at successive time intervals from $t = 0$ to $t = t_f$. It builds a loop counter of the form .fffcc by prompting you for the value of t_f, and for the value of *STEP* (the value of the time interval). t_f becomes the fff portion of the counter and *STEP* becomes the ii portion of the counter.

Here is a flowchart for DISPL.



The program segment that uses a controlled loop to calculate successive values of x is highlighted in the following annotated listing.

To key in DISPL: Create variables *x*, *x0*, *v*, *tF*, *STEP*, *fff*, *ii*, and *COUNT* before program entry.

Program: / **Comments:**

```
00 { 110-Byte Prgm }
01 LBL "DISPL"

02 SF 21
```

```
03 INPUT "x0"
04 INPUT "v"
05 INPUT "tF"
06 INPUT "STEP"
07 RCL "tF"
08 1E-3
09 ×
10 STO "fff"
11 RCL "STEP"
12 1E-5
13 ×
14 STO "ii"
15 RCL+ "fff"
16 STO "COUNT"
```

Lines 03 – 16: Prompt for the variables. Build the counter.

```
17 LBL 01
18 RCL "COUNT"
19 IP
20 RCL× "v"
21 RCL+ "x0"
22 STO "x"
23 CLX
24 VIEW "x"
25 ISG "COUNT"
26 GTO 01
27 GTO "DISPL"
```

Lines 17 – 27: Calculate successive values of *x* in the counter-controlled loop. (Note that the integer part of *COUNT* in line 19 is the time *t*.)

```
28 END
```

Example: Loop Control in a Program. Find successive values of the displacement x of an object in intervals of five seconds from $t = 0$ to $t = 15$ seconds when $x_0 = 10$ meters and $v = 20$ meters/second.

Begin program execution.

[XEQ] DISPL

```
Y: 0.0000
x0?0.0000
```

Enter the values for x_0 and v.

10 [R/S] 20 [R/S]

```
Y: 20.0000
tF?0.0000
```

Enter the value for t_f and continue program execution.

15 [R/S]

```
Y: 15.0000
STEP?0.0000
```

Enter the value of *STEP* (the size of the interval) and continue program execution.

5 [R/S]

```
x=10.0000
x: 0.0000
```

The value of x at $t = 0$ is 10. Press [R/S] again to display the value of x at $t = 5$.

[R/S]

```
x=110.0000
x: 0.0000
```

Press [R/S] to see the value of x at $t = 10$.

[R/S]

```
x=210.0000
x: 0.0000
```

Press [R/S] again to see the value of x at $t = 15$.

[R/S]

```
x=310.0000
x: 0.0000
```

Press [R/S] again to prompt for new values. Exit from the program.

[R/S] [EXIT]

```
Y: 0.0000
x: 10.0000
```

Indirect Addressing in Programs

Indirect addressing is a useful programming tool, particularly when used in combination with a controlled loop. The operation index in your owner's manual indicates which functions can use indirect addresses. In this section, three applications of indirect addressing in programs are presented.

Using Indirect Addressing to Initialize Data Storage Registers. Program INIT prompts for data and stores it in successive registers using INPUT IND in a controlled loop. This is a useful initialization routine if you are using registers instead of variables for data storage and recall.

```
00 { 37-Byte Prgm }
01 LBL "INIT"
```

```
02 1.01
03 STO "COUNT"
```

Lines 02–03: Build a counter and store it in *COUNT*. The counter has a beginning value of 1, a test value of 10, and a default increment value of 1.

```
04 LBL 01
05 INPUT IND "COUNT"
06 ISG "COUNT"
07 GTO 01
```

Lines 04–07: Prompt for data for successive registers $R_{01} - R_{10}$.

```
08 END
```

Using Indirect Addressing to Clear Registers. The following routine clears a specified number of storage registers using STO IND in a controlled loop.

Program:	**Comments:**
`00 { 74-Byte Prgm }` `01 LBL "CLEAR"`	
`02 0`	Line 02: Initialize the X-register to 0.

`03 "FIRST?"`
`04 PROMPT`
`05 STO "COUNT"`
`06 "LAST?"`
`07 PROMPT`
`08 1E-3`
`09 ×`
`10 STO+ "COUNT"`
	Lines 03 – 10: Build a counter in *COUNT*. The counter has a beginning value equal to the first data storage register to be cleared, a test value equal to the last register to be cleared, and an increment value of one.

`11 LBL 10`
`12 0`
`13 STO IND "COUNT"`
`14 ISG "COUNT"`
`15 GTO 10`
	Lines 11 – 15: Successively set the values of the block of specified registers to 0.

`16 TONE 9`
`17 "READY"`
`18 PROMPT`
	Lines 16 – 18: Sound a tone and display the message `READY`. Press [R/S] to end the program.

`19 END`	

Using Indirect Addressing to Execute Subroutines. The following routine retrieves data (telephone numbers) from subroutines using XEQ IND.

Program:	**Comments:**
`00 { 134-Byte Prgm }` `01 LBL "PHONE"`	

`02 "NAME?"`
`03 AON`
`04 PROMPT`
`05 AOFF`
`06 ASTO ST X`
`07 XEQ IND ST X`
`08 PROMPT`
	Lines 02 – 08: Prompt for the name (Alpha string) whose telephone number is desired (lines 02 – 05) and store the string in the X-register (line 06). (The string may be six Alpha characters maximum; the X-register holds only up to six Alpha characters.) Execute the subroutine whose label matches the Alpha string (line 07), then suspend program execution (line 08).

`09 LBL "JANET"`
`10 "000-555-9874"`
`11 RTN`
`12 LBL "BRUCE"`
`13 "000-555-1356"`
`14 RTN`
`15 LBL "PAM"`
`16 "000-555-6093"`
`17 RTN`
`18 LBL "CHRIS"`
`19 "000-555-6276"`
`20 RTN`
`21 LBL "BOB"`
`22 "000-555-2411"`
`23 RTN`
	Lines 09 – 23: Build the telephone numbers (actually Alpha strings) in the Alpha register.

`24 END`	

Flags in Programs

Earlier in this chapter you wrote a program SSA that makes a branch based on a number test; specifically, SSA uses the X≤Y? function to construct the branch. The program asks the question: *Is* $S_2 \leq S_1$? Then it makes a decision based on the answer – either calculate the second answer set or end the program.

The X?0 and X?Y sets of functions enable programs to ask questions *only* concerning number values.* However, programs can also make conditional branches (ask questions and make decisions) based on *flag* tests. Flag tests follow the "do-if-true" rule. If the test is true, the next instruction is executed. If the test is false, the next instruction is skipped. Because flags have unique meanings for the calculator, they greatly expand the logic control you can exercise in a program. (User flags 00 through 35 and 81 through 99 may be set, cleared and tested. System flags 36 through 80 may only be tested. Refer to appendix C in your owner's manual for a complete listing of the HP-42S flags and their meanings.)

User Flags

Flags 00 through 35 and 81 through 99 are *user* flags; they may be set, cleared, and tested.

General Purpose Flags. *General purpose flags* (flags 00 through 10 and 81 through 99) are not used internally by the calculator; what they mean depends entirely on how *you* define them.

* The X=Y? and X≠Y? functions are exceptions; they can compare Alpha strings.

The program LIST on pages 176 through 178 creates a matrix *ΣLIST* using the following instruction sequence.

```
31 LBL 02
32 1
33 ENTER
34 FC? 01
35 2
36 DIM "ΣLIST"
37 XEQ I
38 R↓
39 R↓
40 GTO 00
```

Before you execute LIST, you *set* flag 01 if you want *ΣLIST* to be a 1-column matrix, or you *clear* flag 01 if you want *ΣLIST* to be a 2-column matrix. Flag 01 is defined to have a unique meaning in the program; its status determines the number of columns in the matrix *ΣLIST*. (Remember that current status of user flags is maintained by HP-42S Continuous Memory. This can affect other programs that use the same flags.)

Control Flags. *Control flags* 11 through 35 have a specific meaning and are used internally by the calculator. For example, flag 21, the Printer Enable flag, affects the way the VIEW and AVIEW functions work in programs. When flag 21 is *set* in PROFF mode, VIEW and AVIEW messages are displayed, and program execution halts. When flag 21 is set and PRON is executed, VIEW and AVIEW messages are printed and program execution does not halt. Many programs in this manual that use VIEW or AVIEW also *set* flag 21.

System Flags

System flags 36 through 80 also have a specific meaning for the calculator. You cannot directly set or clear these flags. However, you can test them.

The following program, MINMAX, searches for the maximum or minimum element of the matrix in the X-register. In line 23, it tests the status of system flag 77, the Matrix End-Wrap flag, to determine if the last element of the matrix has been checked.

MINMAX also uses general purpose flag 09 in line 08 to determine whether to search for the maximum or minimum element of the matrix. Before you execute the program, you set flag 09 to find the maximum element, or clear flag 09 to find the minimum element.

(The annotated listing is on pages 152 through 153.)

```
00 { 61-Byte Prgm }
01 LBL "MINMAX"

02 STO "MINMAX"
03 INDEX "MINMAX"

04 RCLEL
05 GTO 03

06 LBL 01
07 RCLEL
08 FS? 09
09 GTO 02
10 X≥Y?
11 GTO 04
12 GTO 03

13 LBL 02
14 X≤Y?
15 GTO 04

16 LBL 03
17 RCLIJ
18 RCL ST Z
19 ENTER

20 LBL 04
21 R↓
22 J+
23 FC? 77
24 GTO 01

25 END
```

Error Trapping

When you attempt an improper operation during function execution, the operation is not executed and an explanatory message is displayed. For example, if you execute the keystroke sequence

1 [E] 260 ■ [x^2]

the calculator returns the message `Out of Range`, and leaves the value 1×10^{260} in the X-register.

If an improper operation is attempted *in a program*, the calculator returns the corresponding message, and program execution *halts* at the instruction that caused the error. Consider the following program.

```
00 { 26-Byte Prgm }
01 LBL "TRAP"
02 SF 21
03 INPUT "X"
04 X↑2
05 STO "Y"
06 VIEW "Y"
07 GTO "TRAP"
08 END
```

If you execute TRAP and supply the value 1×10^{260} for *X*, the program halts at line 03 and the calculator displays the message `Out of Range`. To supply a new value for *X*, you must *restart* the program at line 01 (by pressing [XEQ] `TRAP`). In a short program like TRAP, this method of recovery from an error presents little problem. However, when executing a program that performs time-consuming calculations, or that has numerous stops for intermediate data entry, it may be inconvenient to restart the program at line 01 each time an error occurs.

You can enable program execution to *continue* after an error has occurred by setting flag 25, the Error Ignore flag. When flag 25 is set:

- *One* error during program execution is ignored. The instruction that causes the error is not performed and program execution continues at the next instruction.
- The error *clears* flag 25.

Consider this revision to TRAP.

```
00 { 58-Byte Prgm }
01 LBL "TRAP"

02 SF 21
03 SF 25
04 INPUT "X"
05 X↑2
06 FC?C 25
07 GTO 00
08 STO "Y"
09 VIEW "Y"
10 GTO "TRAP"

11 LBL 00
12 CF 21
13 BEEP
14 "Out of Range"
15 AVIEW
16 PSE
17 PSE
18 GTO "TRAP"

19 END
```

TRAP now responds to the error condition by:

- Displaying an error message.
- Resetting flag 25 and prompting for a new value for X.

This programming technique, called *error trapping*, adds program steps, but is effective when you can identify operations in a program that are likely to generate errors.

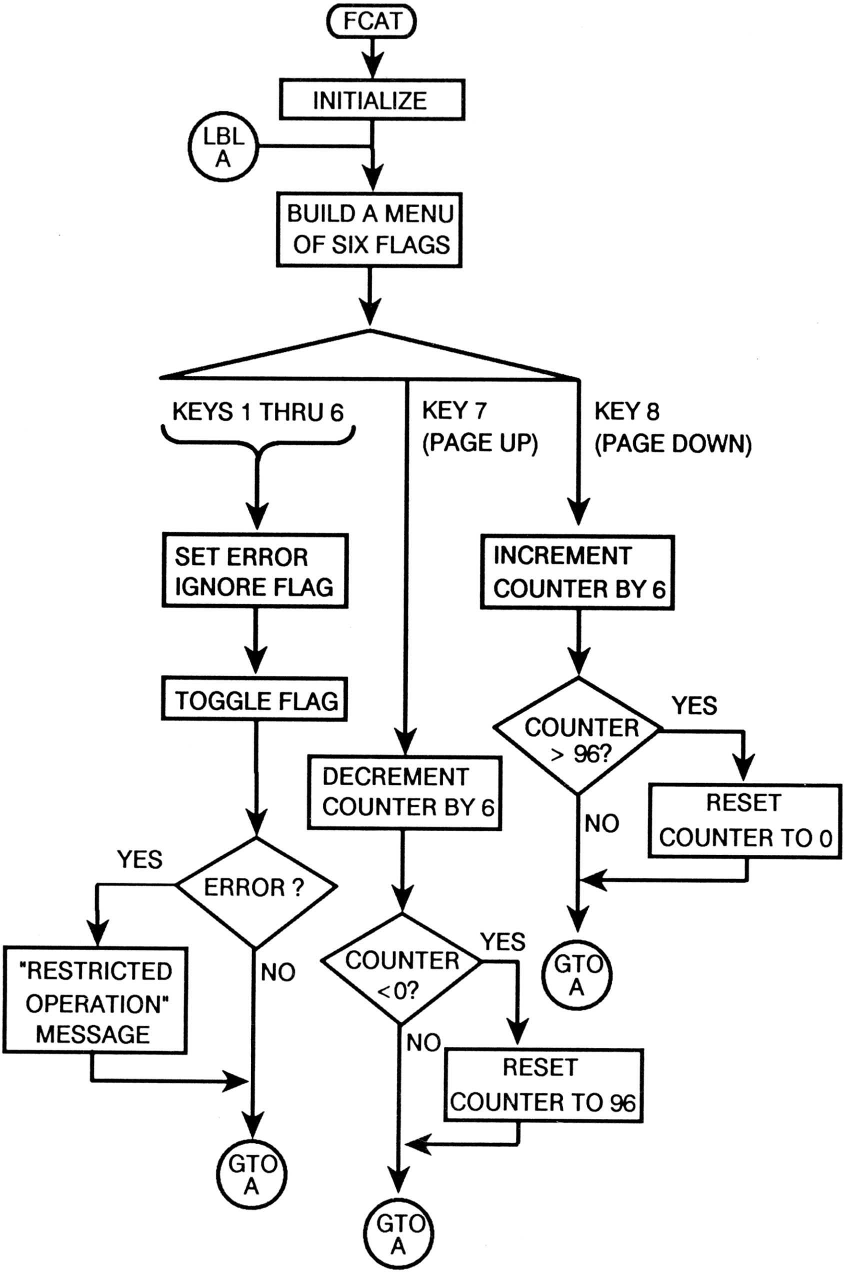
A Summary Program

The program FCAT in this section displays the current status of flags 00 through 99. The flags are displayed in a multirow menu in sets of six. Each of the menu keys is labeled with a flag number. You can set and clear user flags 00 through 35 (except flag 25) and 81 through 99 by pressing the corresponding menu key. The "■" character is appended to the menu label if that flag is currently set. When you attempt to set or clear a system flag, FCAT beeps and displays the error message `Restricted Operation`. The previous set of six flags is displayed by pressing menu key 7 ([▲]), and the succeeding set is displayed by pressing menu key 8 ([▼]).

FCAT uses many of the programming concepts discussed in this chapter:

- Global and local labeling.
- Prompting for data input.
- Conditional branching based on:
  - Number tests.
  - Flag tests.
- Subroutines.
- Multirow menus.
- Counter-controlled looping.
- Indirect addressing.
- Error trapping.

Here is a flowchart for FCAT.



Here is the annotated listing.

Program: / **Comments:**

```
00 { 234-Byte Prgm }
01 LBL "FCAT"
```

```
02 0.09606
03 STO 00
```

Lines 02 – 03: Store the loop counter in R_{00}.

```
04 LBL A
05 RCL 00
06 XEQ 00
07 KEY 1 GTO 01
08 XEQ 00
09 KEY 2 GTO 02
10 XEQ 00
11 KEY 3 GTO 03
12 XEQ 00
13 KEY 4 GTO 04
14 XEQ 00
15 KEY 5 GTO 05
16 XEQ 00
17 KEY 6 GTO 06
```

Lines 4 – 17: Build menu keys 1 – 6. The label for each menu key is built by calling subroutine 00. (Now go to subroutine 00.)

```
18 KEY 7 GTO 07
19 KEY 8 GTO 08
```

Lines 18 – 19: Assign GTO instructions to menu keys 7 and 8.

```
20 "FLAG CATALOG"
21 MENU
22 6
23 STO 01
24 PROMPT
25 GTO A
```

Lines 20 – 25: Build the Alpha string FLAG CATALOG (line 20). Display the menu (line 21). Initialize register R_{01} to 6 (lines 22 – 23). Display the Alpha register, suspend program execution, and prompt for numeric input (line 24).

```
26 LBL 00
27 CLA
28 99.1
29 X<>Y
30 X>Y?
31 RTN
32 AIP
33 FS? IND ST X
34 ├"■"
35 1
36 +
37 RTN
```

Subroutine 00, lines 26–37: Build the Alpha string for each menu key. First, test to see if the current value in the X-register (the loop counter) is greater than 99 (lines 28–31). If yes, do *not* build a label for the menu key. (The highest numbered flag is 99.) If no, append the (integer portion of) the value in the X-register to the Alpha register (line 32). Test the status of the flag whose number is in the X-register. If that flag is *set*, append the "■" character to the Alpha register (lines 33–34). (Thus, the Alpha label for each menu key consists of a number, and, *if the corresponding flag is set*, a "■".) Increment the value of the X-register by 1 (lines 35–36).

```
38 LBL 01
39 DSE 01
40 LBL 02
41 DSE 01
42 LBL 03
43 DSE 01
44 LBL 04
45 DSE 01
46 LBL 05
47 DSE 01
48 LBL 06
49 DSE 01
50 LBL 14
51 RCL 01
52 RCL+ 00
```

Lines 38–52 establish the flag to be set or cleared: Successively decrement R_{01} by 1 (lines 38–49). (If menu key 1 is pressed, the value in R_{01} is 0 when R_{01} is recalled to the X-register in line 51. If menu key 6 is pressed, the value in R_{01} is 5 when R_{01} is recalled to the X-register.) Add the current value in R_{00} (the counter) to the current value in the X-register (line 52). (The value in the X-register after execution of line 52 is the value of the flag to be set or cleared.)

```
53 SF 25
54 FC?C IND ST X
55 GTO 09
56 GTO A
```

Lines 53 – 56 build the set/clear toggle and error trap: Set the Error Ignore flag (line 53). Test if the flag (whose number value is in X) is clear, then clear it (line 54). If the flag was *clear* when tested in line 54, *or* the attempt to clear causes a Restricted Operation error, go to label 09 (line 55). If the flag was *set*, *and* the clear operation does not cause a Restricted Operation error, return to the menu-label routine to update the flag status (line 56).

```
57 LBL 09
58 FC?C 25
59 GTO 10
60 SF IND ST X
61 GTO A
```

Lines 57 – 61: If the branch to label 09 was caused by a Restricted Operation error, go to label 10 (lines 57 – 59). If the branch to subroutine 09 was executed because the flag was clear, then set it, and return to the menu-label routine to update the flag status (lines 60 – 61).

```
62 LBL 07
63 RCL 00
64 6
65 -
66 X<0?
67 96.09606
68 STO 00
69 GTO A
```

Lines 62 – 69: *Decrement* R_{00} by 6. (Thus, when [▼] is pressed, the top-row menu keys are each relabeled with the number that is six *less than* in the previous menu. If R_{00} has the value 12 when [▼] is pressed, R_{00} takes the value 6, and the menu keys are relabeled 6 – 11.) Test if the new value of R_{00} is less than 0. If yes, store 96 in R_{00} (lines 66 – 68). (Menu keys 1 – 4 will be labeled 96 – 99.)

```
70 LBL 08
71 ISG 00
72 GTO A
73 GTO "FCAT"
```

Lines 70–73: *Increment* R_{00} by 6 using the ISG function. (Remember that the number in R_{00} is the loop counter; it has the initial value 0.09906. When [▲] is pressed, the top row menu keys are each relabeled with the number that is six *greater* than in the previous menu. When the counter test value exceeds 96, program execution transfers to FCAT, restoring the counter to its initial value; the menu keys are thus relabeled 0–5.)

```
74 LBL 10
75 FS?C 21
76 GTO 11
77 XEQ 12
78 GTO A
79 LBL 11
80 XEQ 12
81 SF 21
82 GTO A
83 LBL 12
84 BEEP
85 "Restricted "
86 ⊢"Operation"
87 AVIEW
88 PSE
89 RTN

90 END
```

Lines 74–89: Execute the BEEP function, display the Alpha message `Restricted Operation`, and transfer program execution back to label A. If flag 21 is set, clear it before displaying the Alpha message, then reset it. (Program execution continues, redisplaying the flag menu, *and* the status of flag 21 is maintained.)

Example: The Flag Catalog Program. Use FCAT to set flag 01. Check the status of flag 38. Attempt to set or clear it.

Start FCAT.

[XEQ] FCAT

```
FLAG CATALOG
 0 | 1 | 2 | 3 | 4 | 5
```

Set flag 01.

1

```
FLAG CATALOG
 0 | 1■ | 2 | 3 | 4 | 5
```

Check the status of flag 38.

[▼] [▼] [▼]
[▼] [▼] [▼]

```
FLAG CATALOG
 36 | 37■ | 38 | 39 | 40■ | 41
```

Flag 38 is clear. Attempt to set it.

38

```
FLAG CATALOG
 36 | 37■ | 38 | 39 | 40■ | 41
```

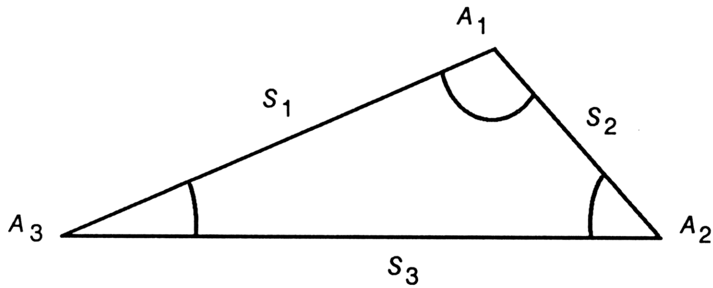
The calculator beeps, displays the message `Restricted Operation`, and returns to the state before the error. Exit from FCAT.

[EXIT]

```
Y: 42.0961
X: 6.0000
```

The Triangle Solutions Program

This section contains the complete set of equations for the triangle solutions, instructions for keying in TRI∡, an annotated listing of TRI∡, and instructions for using TRI∡.



Program Equations. The following equations are used in the program:

- Condition 1: S_1, S_2, and S_3 (three sides) are known:

$$A_3 = 2 \text{ arc cos} \left[\frac{\sqrt{P(P - S_2)}}{(S_1 S_3)} \right] \text{ where } P = \frac{(S_1 + S_2 + S_3)}{2}$$

$$A_2 = 2 \text{ arc cos} \left[\frac{\sqrt{P(P - S_1)}}{(S_2 S_3)} \right]$$

$$A_1 = \text{arc cos} \left[-\cos (A_3 + A_2) \right]$$

- Condition 2: S_1, S_2, and A_2 (two sides and the adjacent angle) are known:

$$A_3 = \text{arc sin}\left[\left(\frac{S_2}{S_1}\right)\sin A_2\right]^*$$

$$A_1 = \text{arc cos}\,[\,-\cos(A_2 + A_3)\,]$$

The problem has been reduced to the A_3, S_1, A_1 configuration.

- Condition 3: A_3, S_1, and A_1 (two angles and the included side) are known:

$$A_2 = \text{arc cos}\,[\,-\cos(A_3 + A_1)\,]$$

$$S_2 = S_1\left(\frac{\sin A_3}{\sin A_2}\right)$$

$$S_3 = S_1 \cos A_3 + S_2 \cos A_2$$

- Condition 4: S_1, A_1, and A_2 (one side and the following two angles) are known:

$$A_3 = \text{arc cos}\,[\,-\cos(A_1 + A_2)]$$

The problem has been reduced to the A_3, S_1, A_1 configuration.

- Condition 5: S_1, S_2 (two sides and the included angle) are known:

$$S_3 = \sqrt{S_1^2 + S_2^2 - 2S_1S_2\cos A_1}$$

The problem has been reduced to the S_1, S_2, S_3 configuration.

- For any triangle, the area is:

$$AREA = \frac{1}{2}S_1S_3\sin A_3$$

* Two possible solutions exist if S_2 is greater than S_1 and A_3 does not equal 90°. Both possible answer sets are calculated.

To key in TRI∡:

1. Create variables *S1*, *S2*, *S3*, *A1*, *A2*, *A3*, *P*, and *AREA* before program entry.
2. Create labels RESULTS and SSASUB when you begin program entry.

Here is an annotated listing of TRI∡.

Program:	**Comments:**
`00 { 573-Byte Prgm }` `01 LBL "TRI∡"`	
`02 SF 21`	

`03 "SSS"`
`04 KEY 1 XEQ A`
`05 "SSA"`
`06 KEY 2 XEQ B`
`07 "ASA"`
`08 KEY 3 XEQ C`
`09 "SAA"`
`10 KEY 4 XEQ D`
`11 "SAS"`
`12 KEY 5 XEQ E`
	Lines 03 – 12: Build the menu key assignments.

`13 MENU`
`14 STOP`
`15 XEQ "RESULTS"`
`16 GTO "TRI∡"`
	Lines 13 – 16: Display the menu keys.

`17 LBL A`
`18 INPUT "S1"`
`19 INPUT "S2"`
`20 INPUT "S3"`
`21 RCL "S1"`
`22 RCL+ "S2"`
`23 RCL+ "S3"`
`24 2`
`25 ÷`
	Subroutine A, lines 17 – 59: Calculate the SSS solution.

```
26 STO "P"
27 X↑2
28 LASTX
29 RCL× "S2"
30 -
31 RCL "S1"
32 RCL× "S3"
33 ÷
34 SQRT
35 ACOS
36 2
37 ×
38 STO "A3"
39 SIN
40 RCL× "S1"
41 STO 00
42 RCL "P"
43 X↑2
44 LASTX
45 RCL× "S1"
46 -
47 RCL÷ "S2"
48 RCL÷ "S3"
49 SQRT
50 ACOS
51 2
52 ×
53 STO "A2"
54 RCL+ "A3"
55 COS
56 +/-
57 ACOS
58 STO "A1"
59 RTN
```

Subroutine B, lines 60 – 100: Calculate the SSA solution.

```
60 LBL B
61 INPUT "S1"
62 INPUT "S2"
63 INPUT "A2"
```

```
64 SIN
65 RCL× "S2"
66 RCL÷ "S1"
67 ASIN
68 STO "A3"
69 SIN
70 RCL× "S1"
71 STO 00
72 XEQ "SSASUB"
73 RCL "S1"
74 RCL "S2"
75 X≤Y?
76 RTN
77 XEQ "RESULTS"
78 RCL "A3"
79 COS
80 +/-
81 ACOS
82 STO "A3"
83 XEQ "SSASUB"
84 RTN
85 LBL "SSASUB"
86 RCL "A3"
87 RCL+ "A2"
88 COS
89 +/-
90 ACOS
91 STO "A1"
92 RCL "A2"
93 COS
94 RCL× "S2"
95 RCL "A3"
96 COS
97 RCL× "S1"
98 +
99 STO "S3"
100 RTN
```

Subroutine C, lines 101 – 126: Calculate the ASA solution.

```
101 LBL C
102 INPUT "A3"
103 INPUT "S1"
104 INPUT "A1"
105 RCL "A3"
106 RCL+ "A1"
107 COS
108 +/-
109 ACOS
110 STO "A2"
111 RCL "A3"
112 RCL "S1"
113 →REC
114 X<>Y
115 STO 00
116 RCL "A2"
117 1
118 →REC
119 R↓
120 ÷
121 STO "S2"
122 R↑
123 ×
124 +
125 STO "S3"
126 RTN
```

Subroutine D, lines 127 – 150: Calculate the SAA solution.

```
127 LBL D
128 INPUT "S1"
129 INPUT "A1"
130 INPUT "A2"
131 RCL+ "A1"
132 COS
133 +/-
134 ACOS
135 STO "A3"
136 RCL "S1"
```

```
137 →REC
138 X<>Y
139 STO 00
140 RCL "A2"
141 1
142 →REC
143 R↓
144 ÷
145 STO "S2"
146 R↑
147 ×
148 +
149 STO "S3"
150 RTN
```

Subroutine E, lines 151 – 194: Calculate the SAS solution.

```
151 LBL E
152 INPUT "S1"
153 INPUT "A1"
154 INPUT "S2"
155 RCL "A1"
156 X<>Y
157 →REC
158 RCL "S1"
159 -
160 →POL
161 STO "S3"
162 RCL+ "S1"
163 RCL+ "S2"
164 2
165 ÷
166 STO "P"
167 X↑2
168 LASTX
169 RCL× "S2"
170 -
171 RCL "S1"
172 RCL× "S3"
173 ÷
174 SQRT
```

```
175 ACOS
176 2
177 ×
178 STO "A3"
179 SIN
180 RCL× "S1"
181 STO 00
182 RCL "P"
183 X↑2
184 LASTX
185 RCL× "S1"
186 -
187 RCL÷ "S2"
188 RCL÷ "S3"
189 SQRT
190 ACOS
191 2
192 ×
193 STO "A2"
194 RTN
```

```
195 LBL "RESULTS"
196 RCL 00
197 RCL× "S3"
198 2
199 ÷
200 STO "AREA"
201 VIEW "S1"
202 VIEW "A1"
203 VIEW "S2"
204 VIEW "A2"
205 VIEW "S3"
206 VIEW "A3"
207 VIEW "AREA"
208 RTN
```

Subroutine RESULTS, lines 195–208: Calculate *AREA* and display the initial known values and the results.

```
209 END
```

To use TRI∡:

1. Press [XEQ] TRI∡ .
2. Select a solution by pressing the corresponding menu key.
3. Input values as prompted. You can name any side S_1. A_1 is the adjacent angle. You can enter values in a clockwise or counterclockwise order. The values are displayed in the same order as they were entered.

2

Enhancing HP-41 Programs

In chapter 11 of your owner's manual, you keyed in and executed a program originally written for the HP-41 calculator. That program, named QUAD, solves for (real number) roots of quadratic equations. Two programs Q2 and Q3 in this chapter use HP-42S features and functions to enhance QUAD. A third program QSHORT uses only 11 lines to solve for quadratic equation roots.

Using Named Variables

In the HP-42S, data may be stored in and recalled from data storage registers *or* named variables. Programs that use named variables for data storage and recall can be easier to write and read.

In QUAD, the values of coefficients a, b, and c are stored in and recalled from data storage registers. In Q2 these values are stored in and recalled from *named variables* a, b, and c. (Q2 also stores the values of the two roots r_1 and r_2 in named variables *R1* and *R2*. In QUAD, these values are calculated and displayed, but not saved.)

Using HP-42S Data Input and Output Functions

Prompting for Data with INPUT

The HP-42S INPUT function enables programs to prompt for data in *one* program line.

QUAD prompts for the value of a, then stores the value $2a$ in a data storage register with the three-instruction sequence

```
02 "a=?"
03 PROMPT
06 STO 00
```

Q2 uses INPUT (and the named variable a) to replace these three instructions with one.

```
09 INPUT "a"
```

Displaying Data with VIEW

The HP-42S VIEW function enables programs to display data in *one* program line.

QUAD displays the labeled value of r_1 with the three-instruction sequence

```
29 "ROOTS="
30 ARCL X
31 AVIEW
```

Q2 uses VIEW (and the named variable *R1*) to replace these three instructions with one.

```
33 VIEW "R1"
```

Operations with HP-42S Data Types

Programs written for the HP-41 calculators can operate on only two data types: real numbers and Alpha strings. Programs for the HP-42S, however, can also operate on complex numbers and matrices.

In QUAD, complex-number roots cannot be calculated; instead, if the value $b^2 - 4ac$ is less than 0, the calculation is halted and the message `ROOTS COMPLEX` is displayed. In Q2, complex number roots are calculated, stored in variables, and displayed.

Using the Two-Line Display

Programs can effectively show longer messages in the HP-42S two-line display. In Q2, the two-line message

```
Zero Input Invalid.
Press R/S to continue.
```

is displayed if 0 is supplied for variables *a* or *c*.

To key in Q2: Create variables *a*, *b*, *c*, *R1*, and *R2* before program entry.

Here is an annotated listing of Q2.

Program:

```
00 { 132-Byte Prgm }
01 LBL 00
02 "Zero Input Inva"
03 ⊢"lid.␊Press R/S"
04 ⊢" to continue."
05 PROMPT
```

Comments:

Lines 01 – 05: Display the 0-input error message.

```
06 LBL "Q2"
07 CPXRES
08 SF 21
09 INPUT "a"
10 X=0?
11 GTO 00
12 INPUT "b"
13 INPUT "c"
14 X=0?
15 GTO 00
```

Lines 06–15: Set the program to calculate complex numbers, prompt for the values of a, b, and c, and test if 0 is supplied for a or c. (Flag 21 is set in line 08 so that VIEW results are *displayed* in PROFF mode, or *printed* if PRON has been executed.)

```
16 RCL "b"
17 +/-
18 ENTER
19 X↑2
20 4
21 RCL× "a"
22 RCL× "c"
23 -
24 SQRT
```

Lines 16–24: Calculate

$$\sqrt{b^2 - 4ac}$$

```
25 RCL "b"
26 SIGN
27 ×
28 -
29 2
30 ÷
31 RCL÷ "a"
```

Lines 25–31: Calculate either

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

or

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

depending on the sign of b. Lines 25–27 ensure that the root that has the greatest absolute value is calculated first. This improves the accuracy of the results.

```
32 STO "R1"
33 VIEW "R1"
```

Lines 32–33: Store the calculated value in *R1* and display *R1*.

```
34 RCL "c"
35 RCL÷ "a"
36 RCL÷ "R1"
37 STO "R2"
38 VIEW "R2"
```

Lines 34–38: Calculate the second root, store the value in *R2*, and display *R2*.*

```
39 GTO "Q2"
```

Line 39: Return program execution to label Q2.

```
40 END
```

Using Menu Variables

Q2 uses the INPUT function to prompt for the values of the program variables *a*, *b*, and *c*. Q3 uses a *variable menu* to prompt for these values. The corresponding program lines are highlighted in the following annotated listing.

* The quadratic equation $ax^2 + bx + c = 0$ can be divided by a (since a cannot equal 0) yielding $x^2 + \frac{bx}{a} + \frac{c}{a} = 0$. This equation can be factored as $(x - R_1)(x - R_2)$ where R_1 and R_2 are the roots of the equation. By definition of the factoring process, $(R_1)(R_2) = \frac{c}{a}$. Therefore, $R_2 = \frac{c}{(aR_1)}$.

To key in Q3: Create variables *a*, *b*, *c*, *R1*, and *R2* before program entry.

Program:	**Comments:**

```
00 { 143-Byte Prgm }
01 LBL 00
02 "Zero Input Inva"
03 ⊢"lid.␊Press R/S"
04 ⊢" to continue."
05 PROMPT

06 LBL "Q3"
07 MVAR "a"
08 MVAR "b"
09 MVAR "c"
10 CPXRES
11 SF 21
12 VARMENU "Q3"
13 STOP

14 RCL "a"
15 X=0?
16 GTO 00
17 RCL "c"
18 X=0?
19 GTO 00

20 RCL "b"
21 +/-
22 ENTER
23 X↑2
24 4
25 RCL× "a"
26 RCL× "c"
27 -
28 SQRT

29 RCL "b"
30 SIGN
31 ×
```

Lines 06 – 13: Declare menu variables *a*, *b*, and *c*, set the program to calculate complex numbers, set flag 21, and display the variable menu.

```
32 -
33 2
34 ÷
35 RCL÷ "a"

36 STO "R1"
37 VIEW "R1"

38 RCL "c"
39 RCL÷ "a"
40 RCL÷ "R1"
41 STO "R2"
42 VIEW "R2"

43 GTO "Q3"

44 END
```

Assigning a Program to the CUSTOM Menu

When you created the global label Q3 in program line 06, that label was automatically placed in the HP-42S program catalog. You can now execute Q3 by pressing

[XEQ] Q3

(requiring a minimum of two keystrokes, depending on where label Q3 is in the program catalog).

Alternately, you can assign Q3 to the CUSTOM menu by pressing

■ [ASSIGN] PGM Q3

then selecting the desired row of the menu and pressing the desired menu key in that row. The program now can be executed directly from the CUSTOM menu with one keystroke.

Example: Executing an Enhanced HP-41 Program from the CUSTOM Menu.

Part 1. Execute Q3 from the CUSTOM menu to find the roots of the equation

$$x^2 + 6x + 1 = 0\,(a = 1, b = 6, c = 1)$$

Assign Q3 to the CUSTOM menu using the keystroke sequence just described. If you want to print the results, execute PRON. Start the program from the CUSTOM menu.

([PRINT] [▲] PON)
[CUSTOM] Q3

x: 0.0000
A B C

Enter the values for *a*, *b*, and *c*. Then calculate *R1*. (If you are printing the results, you won't see this display.)

1 A
6 B
1 C
[R/S]

R1=-5.8284
A B C

R1 is calculated and displayed. Now check *R2*.

[R/S]

R2=-0.1716
A B C

Return to the start of the program for new data.

[R/S]

x: -0.1716
A B C

Part 2. Find the complex roots of the equation

$$2x^2 + x + 3 = 0\ (a = 2, b = 1, c = 3)$$

Set the angular mode to Rectangular. Enter the values for *a*, *b*, and *c*. Then calculate *R1*. (If you are printing the results, you won't see this display.)

```
■ [MODES] RECT          R1=-0.2500 -i1.1990
2 A                     A B C
1 B
3 C
[R/S]
```

R1 is calculated and displayed. Now check *R2*.

```
[R/S]                   R2=-0.2500 i1.1990
                        A B C
```

Exit from Q3.

```
[EXIT]                  Y: -0.2500 -i1.1990
                        X: -0.2500 i1.1990
```

A Short Quadratic Program. In conclusion, here is an 11-line, 26-byte quadratic equation solver.

```
00 { 26-Byte Prgm }
01 LBL "QSHORT"

02 -0.5
03 x
04 ENTER
05 ENTER
06 X↑2
07 RCL- ST T
08 SQRT
09 STO+ ST Z
10 -

11 END
```

To use QSHORT:

1. Set the calculator to Rectangular mode and to Complex Results mode.
2. Key in the value $\frac{c}{a}$, then press [ENTER].
3. Key in the value $\frac{b}{a}$.
4. Press [XEQ] QSHO.

3

The Solver

The material in this chapter builds on concepts introduced to you in chapter 12 of your owner's manual.

The following topics are covered:

- Basic use of the Solver.
- Providing initial guesses for the Solver.
- Emulating the Solver.
- Using the Solver in programs.
- More on how the Solver works.

Basic Use of the Solver

The general procedure for executing the Solver is:

1. Create a program that:
   a. Uses MVAR to define the variable(s) in the equation.
   b. Expresses the equation such that its right side equals 0. (Note that each variable in the equation must be *recalled* to the X-register.)
2. Apply the Solver to the program:
   a. Press ■ [SOLVER].
   b. Select the program by pressing the corresponding menu key.
   c. Enter the value for each known variable by keying in the value, then pressing the corresponding menu key.
   d. Optional: Supply one or two guesses for the unknown variable by keying in the guess(es), then pressing the corresponding menu key.

e. Find the value of the unknown variable by pressing the corresponding menu key.

Example: Basic Use of the Solver. The equation of state for an ideal gas is

$$PV = nRT$$

where:

P is the pressure of the gas (in atmospheres).
V is the volume of the gas (in liters).
n is the weight of the gas (in moles).
R is the universal gas constant (0.082057 liter-atmosphere/Kelvin-mole).
T is the temperature of the gas (in Kelvins).

Part 1. Create a program for the Solver that declares the variables and expresses the equation.

First, set the right side of the equation equal to 0.

$$PV - nRT = 0$$

Now write the program.

Program:	**Comments:**
`00 { 42-Byte Prgm }` `01 LBL "GAS"`	

`02 MVAR "P"`
`03 MVAR "V"`
`04 MVAR "n"`
`05 MVAR "T"`
	Lines 02–05: Declare the variables.

`06 RCL "P"`
`07 RCL× "V"`
`08 RCL "n"`
`09 RCL× "T"`
`10 0.082057`
`11 ×`
`12 -`
	Lines 06–12: Express the equation such that its right side equals 0.

`13 END`	

Part 2. Use the Solver to find the solution to the following problem.

Calculate the pressure exerted by .305 mole of oxygen in .950 liter at 150 °C (423 K), assuming ideal gas behavior.

Select the Solver application.

■ [SOLVER]

```
Select Solve Program
GAS
```

Select the program you just created.

GAS

```
x: 0.0000
P  V  N  T
```

Enter the values for the variables you know.

.95 V
.305 N
423 T

```
T=423.0000
P  V  N  T
```

Solve for the pressure.

P

```
P=11.1438
P  V  N  T
```

Part 3. Given the same volume and weight of oxygen, what is the temperature of the gas at a pressure of 15 atmospheres?

Since the values of the volume and weight are unchanged, you need only enter the value of the pressure.

15 P

```
P=15.0000
P  V  N  T
```

Now solve for the temperature.

T

```
T=569.3763
P  V  N  T
```

Exit from the Solver application.

[EXIT] [EXIT]

```
y: 569.3763
x: 569.3763
```

Providing Initial Guesses for the Solver

For certain functions, it helps to provide one or two initial guesses for the unknown variable. This can speed up the calculation, direct the Solver to a realistic solution, and find more than one solution, if appropriate.

Directing the Solver to a Realistic Solution

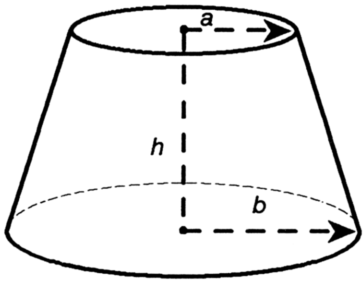
Often, the Solver equation that describes a system may have solution(s) that are mathematically valid but that do not have physical significance. In these cases, it may be necessary to direct the Solver to the realistic solution by providing appropriate initial guesses.

Example: Directing the Solver to a Realistic Solution. The volume of the frustum of a right circular cone is found by

$$V = \frac{1}{3}\pi h\,(a^2 + ab + b^2)$$

where:

V is the volume of the frustum.
h is the height of the frustum.
a is the radius at the top of the frustum.
b is the radius at the base of the frustum.



Part 1. Write a Solver program that declares the variables and expresses the equation such that its right side equals 0.

```
00 { 45-Byte Prgm }
01 LBL "CONE"

02 MVAR "V"
03 MVAR "h"
04 MVAR "a"
05 MVAR "b"

06 RCL "a"
07 X↑2
08 LASTX
09 RCL× "b"
10 +
11 RCL "b"
12 X↑2
13 +
14 RCL× "h"
15 PI
16 ×
17 3
18 ÷
19 RCL- "V"

20 END
```

For the purposes of this example, assume that you have already created variable ***a*** and used it in a previous program. Assume that the value −3.7765 is currently stored in ***a***. (Go ahead now and store that value in ***a*** by pressing 3.7765 [+/-] [STO] A .)

Part 2. For a frustum of volume $V = 119.381$ meters^3, height $h = 6$ meters, and radius b at the base of the cone = 3 meters, use the Solver to find radius a.

Select the Solver application and then program CONE.

■[SOLVER] CONE

```
x: -3.7765
 V  H  A  B
```

Enter the values for the known variables.

119.381 V
6 H
3 B

```
b=3.0000
 V  H  A  B
```

Solve for a.

A

```
a=-5.0000
 V  H  A  B
```

The Solver uses the current value of variable a (−3.7765) as an initial guess and finds the solution $a = -5$ meters. The answer is *mathematically valid*. However, a negative radius clearly has no physical significance. Try guesses of 0 and 5.

0 A
5 A
A

```
a=2.0000
 V  H  A  B
```

The value 2.0000 meters for radius a is mathematically valid and has physical significance.

Exit from the Solver.

[EXIT] [EXIT]

```
Y: 2.0000
x: 2.0000
```

Finding More Than One Solution

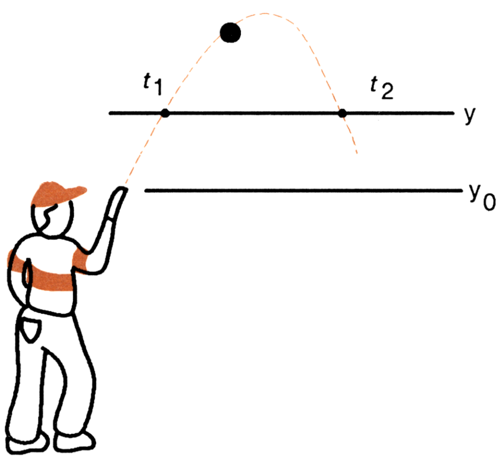
The equation of motion for an object experiencing constant acceleration due to gravity is

$$y = y_0 + v_0 t + \frac{1}{2} g t^2$$

where:

y is the total displacement.
y_0 is the initial position.
v_0 is the initial velocity.
g is the acceleration due to gravity (-9.8 meters/second^2).
t is the time.

In your owner's manual in section "More Solver Examples" in chapter 12, you solved several problems in which an object was *dropped* from an initial position; v_0 was equal to 0 and the direction of the object's motion was down at all times. The object attained a given displacement y at only one time t. However, an object thrown *upwards* attains a given displacement y at *two* different times – once on the way up, and again on the way down.



To find both times t_1 and t_2, you must execute the Solver twice, and at least once provide the Solver with an initial guess to direct it to the second solution.

Example: Using the Solver to Find Two Real Solutions. A boy throws a ball with an initial vertical velocity $v_0 = 15$ meters/second, from an initial height $y_0 = 2$ meters. Use the Solver to find the two times t_1 and t_2 when the ball has a height $y = 5$ meters.

Part 1. Create a Solver program that declares the variables and expresses the equation such that its right side equals 0.

```
00 { 53-Byte Prgm }
01 LBL "MOTION"

02 MVAR "y"
03 MVAR "y0"
04 MVAR "v0"
05 MVARA "t"

06 RCL "y0"
07 RCL "v0"
08 RCL× "t"
09 RCL "t"
10 X↑2
11 -9.8
12 ×
13 2
14 ÷
15 +
16 +
17 RCL- "y"

18 END
```

Part 2. Execute the Solver to find the first time t_1. Since you know that this time is close to 0 seconds, provide initial guesses of 0 and 1.

Select the Solver application and then program MOTION.

[SOLVER] MOTIO

```
x: 0.0000
 Y | Y0 | V0 | T |  |
```

Enter the values for the known variables.

5 Y
2 Y0
15 V0

```
v0=15.0000
 Y | Y0 | V0 | T |  |
```

Solve for time t_1 using initial guesses of 0 and 1.

0 T
1 T
T

```
t=0.2151
 Y | Y0 | V0 | T |  |
```

The Solver finds the value of t_1 = 0.2151 seconds. Now find the second time t_2 by providing two initial guesses that you can expect to bound the second solution. Guesses of 1 and 20 seem reasonable. (You need not enter values for the other variables since they have not changed.)

1 T
20 T
T

```
t=2.8461
 Y | Y0 | V0 | T |  |
```

The Solver finds the value of t_2 = 2.8461 seconds.

Exit from the Solver.

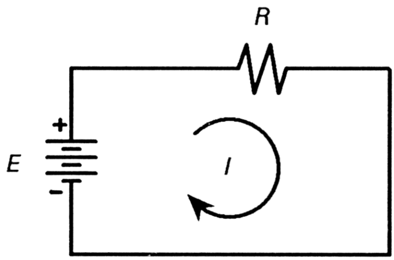
[EXIT] [EXIT]

```
Y: 2.8461
x: 2.8461
```

Emulating the Solver in a Program

For certain types of functions, the Solver algorithm cannot find solutions. For example, the Solver cannot solve for complex numbers. However, for such functions, you can write a program that finds *explicit* solutions and *acts* like the Solver during program execution.

First, consider the following simple circuit.



Ohm's law defines the relationship between the voltage potential E, resistance R, and current I for this circuit as

$$E = IR$$

Since there are no complex terms in this equation, the Solver can be used to find the value of any variable in the equation.

Example: Using the Solver for a Simple Resistive Circuit. For a simple resistive circuit, use the Solver to find the resistance R when the voltage $E = 10$ V, and the current $I = 5$ A.

First, create a Solver program that declares the variables and expresses the Ohm's law equation such that its right side equals 0.

```
00 { 29-Byte Prgm }
01 LBL "CIRCUIT"

02 MVAR "E"
03 MVAR "I"
04 MVAR "R"

05 RCL "I"
06 RCL× "R"
07 RCL- "E"

08 END
```

Select the Solver application and then program CIRCUIT.

■ [SOLVER] CIRCU

```
x: 0.0000
 E  I  R
```

Enter the known values for E and I, then solve for R.

10 E
5 I
R

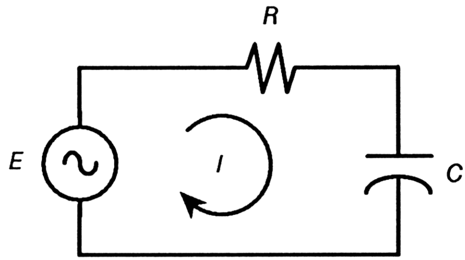
```
R=2.0000
 E  I  R
```

Exit from the Solver application.

[EXIT] [EXIT]

```
y: 2.0000
x: 2.0000
```

Now consider the following circuit.



Application of Ohm's law to this circuit results in the following expression.

$$E = IZ$$

where:

E is the circuit voltage.
I is the circuit current.
Z is the circuit impedance.

The *impedance* Z is the complex number (in rectangular form)

$$R - i\left(\frac{1}{\omega C}\right)$$

where:

R is the circuit resistance.
ω is the circuit frequency (in radians/second).
C is the circuit capacitance.

Because the voltage, current, and impedance are complex numbers, you cannot use the Solver to find their values. However, the HP-42S can perform *arithmetic* operations on complex numbers. (Refer to chapter 6 in your owner's manual for a discussion of complex-number arithmetic.) The following program, EIZ, solves explicitly (algebraically) for the complex numbers E, I, and Z, and uses a *variable menu* to simulate the external appearance of the Solver. (Refer to the section "Using a Variable Menu" in chapter 9 of your owner's manual for a discussion of variable menus.)

Here is an annotated listing of the program.

Program: / **Comments:**

```
00 { 96-Byte Prgm }
01 LBL "EIZ"
```

```
02 MVAR "E∡"
03 MVAR "I∡"
04 MVAR "Z∡"
05 VARMENU "EIZ"
```

Lines 02–05: Declare variables *E*, *I*, and *Z* and build the variable menu.

```
06 POLAR
07 CPXRES
08 CLA
09 STOP
10 ALENG
11 X=0?
12 GTO "EIZ"
```

Lines 06–12: Set the calculator to Polar mode and to calculate complex results. Suspend program execution for data entry. If a variable to solve for has not been specified, return to the start of the program.

```
13 ASTO ST X
14 XEQ IND ST X
15 STO IND ST Y
16 VIEW IND ST Y
17 GTO "EIZ"
```

Lines 13–17: Recall the current Alpha string to the X-register and execute the corresponding subroutine. (The current Alpha string is the name of the variable for which no value is supplied.) Store the calculated result from the subroutine in the Y-register and view the result. Then return to the start of the program.

```
18 LBL "E∡"
19 RCL "I∡"
20 RCL× "Z∡"
21 RTN
```

Subroutine E∡, lines 18–21: Calculate *E*∡ in terms of *I*∡ and *R*∡.

```
22 LBL "I∡"
23 RCL "E∡"
24 RCL÷ "Z∡"
25 RTN
```

Subroutine I∡, lines 22–25: Calculate *I*∡ in terms of *E*∡ and *Z*∡.

```
26 LBL "Z∡"
27 RCL "E∡"
28 RCL÷ "I∡"
29 RTN
```

Subroutine Z∡, lines 26–29: Calculate *Z*∡ in terms of *E*∡ and *I*∡.

```
30 END
```

(Line 06 sets the calculator to Polar mode. Multimeters typically display complex voltage, current, and impedance values in polar form, that is, as a magnitude and phase angle.)

Example: Calculating Complex Values In an RC Circuit. A 10-volt power supply at phase angle 0° drives an RC circuit at a frequency of 40 radians per second. A current of .37 A at phase angle 68° is measured. What is the resistance of the circuit? What is the capacitance of the circuit?

Begin program EIZ.

[XEQ] EIZ

```
x: 0.0000
E∡ | I∡ | Z∡ | | |
```

Enter the known value for the voltage.

10 [ENTER] 0 ■ [COMPLEX] E∡

```
E∡=10.0000 ∡0.0000
E∡ | I∡ | Z∡ | | |
```

Enter the known value for the current.

.37 [ENTER] 68 ■ [COMPLEX] I∡

```
I∡=0.3700 ∡68.0000
E∡ | I∡ | Z∡ | | |
```

Solve for the impedance.

Z∡

```
Z∡=27.0270 ∡-68.0000
E∡ | I∡ | Z∡ | | |
```

The impedance of the circuit (in polar form) is 27 Ω at phase angle −68°. Convert the impedance to rectangular form to find the circuit resistance and capacitance. (Remember, *R* is the real term and *C* is one factor in the imaginary term of the *rectangular* form of the impedance *Z*.)

■ [MODES] RECT

```
x: 10.1245 -i25.0590
EΔ  IΔ  ZΔ
```

The circuit resistance is 10 Ω. Now calculate the capacitance.

■ [COMPLEX]
[+/-] 40 [×]
■ [TOP.FCN] 1/x

```
x: 0.0010
EΔ  IΔ  ZΔ
```

The circuit capacitance is .001 F.

If, at the original input voltage, the impedance is now varied and measures 20 Ω at phase angle −45°, what is the current?

Return to polar mode. Then enter the new value for the impedance and solve for the current.

■ [MODES] POLAR
20 [ENTER] 45 [+/-]
■ [COMPLEX] ZΔ
IΔ

```
IΔ=0.5000 ∡45.0000
EΔ  IΔ  ZΔ
```

The current is 0.5 A at phase angle 45°.

Exit from EIZ.

[EXIT]

```
Y: "IΔ"
x: 0.5000 ∡45.0000
```

Using the Solver in Programs

Using the Solver and Explicit Solutions in a Program

The Solver uses an iterative method to find solutions for the variables in an equation. You *must* use an iterative method to find the solution for a variable that cannot be isolated (cannot be expressed uniquely in terms of the other variables in the equation). However, in cases where the unknown variable can be isolated by algebraic manipulation, an explicit solution for that variable is *always faster* than an iterative solution using the Solver.

Some functions may contain a variable whose value must be found iteratively, and other variables whose values can be calculated explicitly. In your owner's manual, in the section "More Solver Examples" in chapter 12, you worked an example in which the Solver was used to find the solutions to time-value-of-money (TVM) problems. The TVM equation is

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

where:

- N is the number of compounding periods or payments.
- i is the decimal form of the periodic interest rate.
- PV is the present value. (This can also be an initial cash flow or the discounted value of a series of future cash flows.) PV always occurs at the beginning of the first period.
- PMT is the periodic payment.
- FV is the future value. (This can also be a final cash flow or the compounded value of a series of cash flows.) It always occurs at the end N^{th} period.
- p is the payment timing. If $p = 1$, payments occur at the *beginning* of the period. If $p = 0$, payments occur at the *end* of the period.

In the example in your owner's manual, you wrote a program TVM that declares each of the TVM variables and expresses the TVM equation. The Solver is used to find the solution for each of the function variables. Notice, though, that the variables *PV*, *N*, *FV*, and *PMT* can each be isolated. For example, *PV* can be expressed as

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

Only the variable i cannot be isolated; you need to use the Solver only when you want to find the value of i.

The following program, TVM2, calculates the solutions to *PV*, *N*, *FV*, and *PMT* explicitly, and calls the Solver to find the solution for i. The program uses a programmable menu and flag 22, the Numeric Data Input flag, to simulate the external appearance of the Solver application.

To key in TVM2: Create variables *P/YR*, *p*, *CNTRL*, *N*, *FV*, *MODE*, *PMT*, *i*, *I%YR*, and *PV*.

Here is an annotated listing.

Program: / **Comments:**

```
00 { 533-Byte Prgm }
01 LBL "TVM2"
```

```
02 REALRES
03 CF 21
04 12
05 SF 25
06 RCL "P/YR"
07 XEQ 21
08 SF 25
09 RCL "p"
10 CF 25
11 1
12 X≠Y?
13 0
14 STO "p"
15 XEQ 20
```

Lines 02–15: Ensure results are real numbers. Display AVIEW messages and continue program execution. Call subroutine 21 to set the default payments per year to 12. Set the default payment mode to End mode. Call subroutine 20 to display the payments per year and the payment mode.

```
16 LBL 99
17 CLMENU
18 "N"
19 KEY 1 XEQ 01
20 "I%YR"
21 KEY 2 XEQ 02
22 "PV"
23 KEY 3 XEQ 03
24 "PMT"
25 KEY 4 XEQ 04
26 "FV"
27 KEY 5 XEQ 05
28 "MODES"
29 KEY 6 GTO 06
30 MENU
31 STOP
32 ASTO "CNTRL"
33 STO IND "CNTRL"
34 VIEW IND "CNTRL"
35 GTO 99
```

Lines 16 – 35: Build the main menu, display it, and wait for data input (lines 17 – 31). Display the value of the entered or calculated variable (lines 32 – 34).

```
36 LBL 20
37 CLA
38 RCL "P/YR"
39 AIP
40 ⊢" P/YR"
41 RCL "p"
42 X=0?
43 ⊢"   END MODE"
44 X≠0?
45 ⊢"  BEGIN MODE"
46 AVIEW
47 CLMENU
48 RTN
```

Subroutine 20, lines 36 – 48: Build and display the payments-per-year and payment-mode message.

```
49 LBL 06
50 XEQ 20
51 "P/YR"
52 KEY 1 XEQ 21
53 "BEG"
54 KEY 2 XEQ 22
55 "END"
56 KEY 3 XEQ 23
57 "TVM"
58 KEY 4 GTO "TVM2"
59 MENU
60 RCL "P/YR"
61 STOP
62 GTO 06
```

Lines 49 – 62: Build and display the payments-per-year and payment-mode menu.

```
63 LBL 21
64 ABS
65 IP
66 1000
67 X<>Y
68 X≥Y?
69 12
70 X=0?
71 12
72 STO "P/YR"
73 RTN
```

Subroutine 21, lines 63 – 73: Check if the specified number of payments per year is valid. If not, substitute 12 payments per year.

```
74 LBL 22
75 1
76 STO "p"
77 RTN
```

Subroutine 22, lines 74 – 77: Set payment mode to Begin by supplying 1 for *p*.

```
78 LBL 23
79 0
80 STO "p"
81 RTN
```

Subroutine 23, lines 78 – 81: Set payment mode to End by supplying 0 for *p*.

Subroutine 01, lines 82–107: If numeric input is made for *N*, return to the main menu and display the value of *N*. If not, calculate *N* in terms of the other variables. If *i* = 0, go to label 00 to calculate *N* (lines 93–95).

```
82 LBL 01
83 "N"
84 FS?C 22
85 RTN
86 1
87 STO "N"
88 XEQ 10
89 RCL "FV"
90 RCL÷ "MODE"
91 +/-
92 RCL "PMT"
93 RCL "i"
94 X=0?
95 GTO 00
96 ÷
97 +
98 LASTX
99 RCL "PV"
100 RCL÷ "MODE"
101 +
102 ÷
103 LN
104 RCL "i"
105 LN1+X
106 ÷
107 RTN
```

Lines 108–113: Calculate *N* if *i* is 0.

```
108 LBL 00
109 RCL "PV"
110 RCL+ "FV"
111 RCL÷ "PMT"
112 +/-
113 RTN
```

```
114 LBL 02
115 "I%YR"
116 FS?C 22
117 RTN
118 PGMSLV "i"
119 0
120 STO "I%YR"
121 20
122 SOLVE "I%YR"
123 RTN
```

Subroutine 02, lines 114–123: Use the Solver to calculate *I%YR*. Specify the Solver subroutine "i". Supply initial guesses of 0 and 20 for *I%YR*.

```
124 LBL "i"
125 XEQ 10
126 RCL× "PMT"
127 X<>Y
128 RCL× "FV"
129 +
130 RCL+ "PV"
131 RTN
```

Subroutine "i", lines 124–131: Express the TVM equation for the Solver.

```
132 LBL 03
133 "PV"
134 FS?C 22
135 RTN
136 XEQ 10
137 RCL× "PMT"
138 X<>Y
139 RCL× "FV"
140 +
141 +/-
142 RTN
```

Subroutine 03, lines 132–142: If numeric input is made for *PV*, return to the main menu and display the value of *PV*. If not, calculate *PV* in terms of the other variables.

```
143 LBL 04
144 "PMT"
145 FS?C 22
146 RTN
147 XEQ 10
148 X<>Y
```

Subroutine 04, lines 143–154: If numeric input is made for *PMT*, return to the main menu and display the value of *PMT*. If not, calculate *PMT* in terms of the other variables.

```
149 RCL× "FV"
150 RCL+ "PV"
151 X<>Y
152 ÷
153 +/-
154 RTN
```

```
155 LBL 05
156 "FV"
157 FS?C 22
158 RTN
159 XEQ 10
160 RCL× "PMT"
161 RCL+ "PV"
162 X<>Y
163 ÷
164 +/-
165 RTN
```

Subroutine 05, lines 155 – 165: If numeric input is made for *FV*, return to the main menu and display the value of *FV*. If not, calculate *FV* in terms of the other variables.

```
166 LBL 10
167 RCL "I%YR"
168 RCL÷ "P/YR"
169 100
170 ÷
171 STO "i"
172 RCL× "p"
173 1
174 +
175 STO "MODE"
176 1
177 ENTER
178 RCL+ "i"
179 RCL "N"
180 +/-
181 Y↑X
182 STO ST Z
183 -
184 RCL× "MODE"
185 SF 25
186 RCL÷ "i"
```

Subroutine 10, lines 166 – 188: Calculate terms of the TVM equation based on the value of *I%YR*. Calculate *i*; the decimal form of the periodic interest rate (lines 167 – 171). Calculate *MODE* $(1 + ip)$ (lines 172 – 175). Calculate the *FV* coefficient $(1 + i)^{-N}$ (lines 176 – 182). Calculate the *PMT* coefficient. If $i = 0$, go to line 189 (lines 183 – 188).

```
187 FS?C 25
188 RTN
```

```
189 1
190 RCL "N"
191 END
```

Lines 189 – 191: If $i = 0$, then the *FV* coefficient is 1 and the *PMT* coefficient is *N*.

To use TVM2:

1. Press [XEQ] TVM2 .
2. Supply values for the known variables. For example, press 60 N .
3. Solve for the unknown variable by pressing the corresponding menu key.
4. TVM2 uses the variable *I%YR* to prompt for and display the interest rate. *I%YR* is the percent form of the annualized interest rate.
5. The default payment period is one month (12 payments per year). The default payment timing is the *end* of each period. To specify a different payment period or payment timing, first select the MODE menu. Then, for example, to specify six payments per year, press 6 P/YR .

   To specify payment timing at the *beginning* of each period, press BEG .

   To return to the main menu, press TVM .

Example: Executing Algebraic Solutions for TVM Problems. In the section "More Solver Examples" in chapter 12 of your owner's manual, Penny of Penny's Accounting wants to calculate the monthly payment *PMT* for a 3-year loan financed at a 10.5% annual interest rate, compounded monthly. The loan amount is $5,750.

In that example, you executed the program TVM to calculate the value $PMT = -186.89$. TVM *uses the Solver* to calculate *PMT*. The calculation takes about three seconds with initial guesses of 0 and -500.

Part 1. Use TVM2 to calculate the value of *PMT* explicitly.

Set the display format to FIX 2. Then execute TVM2.

■ [DISP] FIX 2 [ENTER]
[XEQ] TVM2

```
12 P/YR    END MODE
N I%YR PV PMT FV MODE
```

Enter the known values.

5750 PV
10.5 I%YR
36 N
0 FV

```
FV=0.00
N I%YR PV PMT FV MODE
```

Solve for the payment.

PMT

```
PMT=-186.89
N I%YR PV PMT FV MODE
```

The explicitly calculated value is −186.89 (the same as when you used TVM) and the calculation takes less than one second. Also note that the calculation time is *independent* of the previously calculated value *PMT*. (The Solver interprets the previously calculated value as a guess if two guesses are not supplied. The explicit solution does not use guesses.)

Part 2. Another bank has offered to loan Penny's customer $5,750, to be paid in monthly installments of $200. What interest rate is this bank charging?

200 [+/-] PMT
I%YR

```
I%YR=15.24
N I%YR PV PMT FV MODE
```

TVM uses the Solver to calculate the new interest rate. The Solver uses the guesses 0 and 20 (supplied by the program) to start its iterative search. The calculation takes about 11 seconds.

Exit from TVM2 and return the display format to FIX 4.

[EXIT]
■ [DISP] FIX 4 [ENTER]

```
Y: 15.2393
X: 15.2393
```

Using the SOLVE and PGMSLV Functions with Indirect Addresses

In the previous section, you used the SOLVE function in TVM2 to find the value of the interest rate *i* in the TVM equation:

```
122 SOLVE "I%YR"
```

You used the PGMSLV function to specify the routine that expresses the TVM equation:

```
118 PGMSLV "i"
```

In TVM2, the SOLVE and PRGSLV instructions *directly* address the variable and the subroutine. Such use of direct addressing enables you to specify only one Solver routine and, within that routine, only one variable. However, the use of *indirect* addressing expands the utility of the Solver by enabling you to specify any of multiple routines, and any of multiple variables.

Example: Using SOLVE with an Indirect Address. Restating the ideal gas equation of state:

$$PV - nRT = 0$$

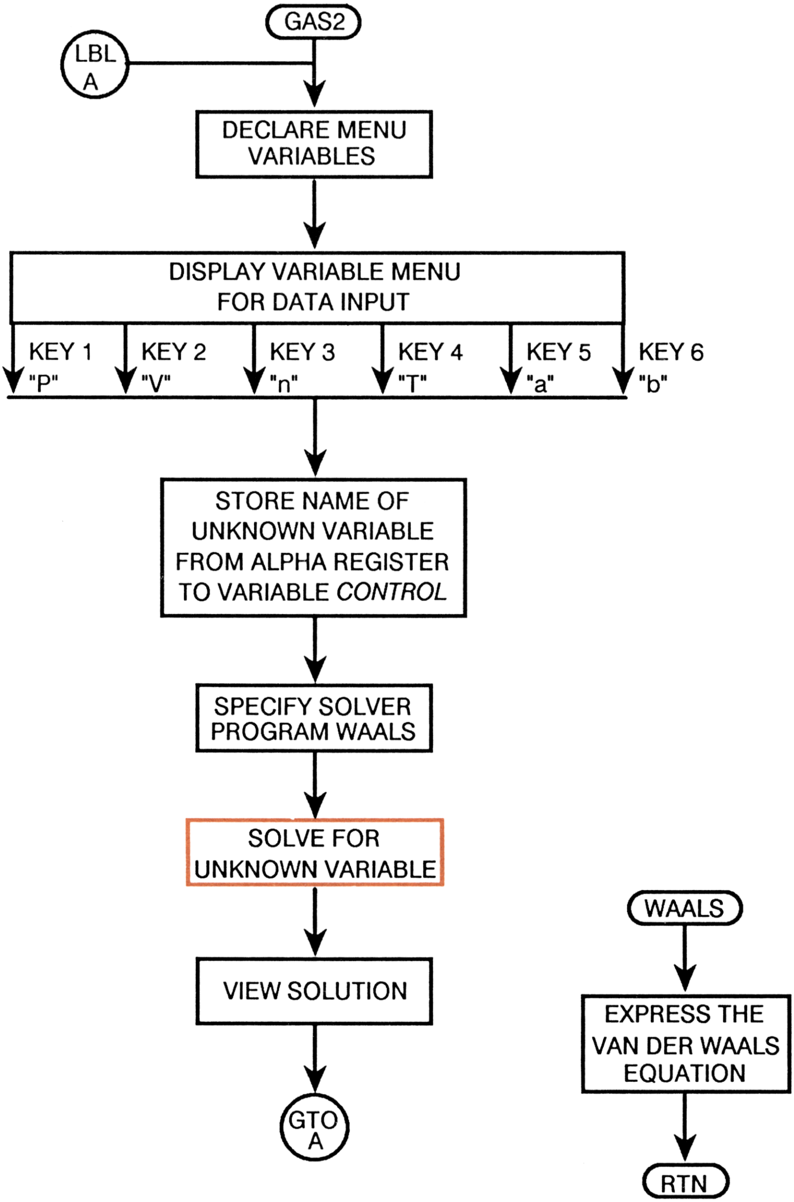
The "van der Waals" equation of state refines the ideal gas equation to

$$\left[\left(P + \frac{n^2 a}{V^2}\right)(V - nb)\right] - nRT = 0$$

where a and b are constants characteristic of the gas in question.

Part 1. Write a program that enables you to solve for the value of any of the variables using either the ideal gas or van der Waals equation of state.

Here is a flowchart for the program, named GAS2.



Here is an annotated listing of the program.

Program: / **Comments:**

```
00 { 129-Byte Prgm }
01 LBL "GAS2"
```

```
02 MVAR "P"
03 MVAR "V"
04 MVAR "n"
05 MVAR "T"
06 MVAR "a"
07 MVAR "b"
08 VARMENU "GAS2"
```

Lines 02 – 08: Build the variable menu.

```
09 CF 21
10 REALRES
11 STOP
12 ASTO "CONTROL"
13 PGMSLV "WAALS"
14 SOLVE IND "CONTROL"
15 VIEW IND "CONTROL"
16 GTO "GAS2"
```

Lines 09 – 16: Clear flag 21 to continue program execution after a VIEW instruction. Set to calculate real results only. Display the menu. Store the name of the unknown variable in *CONTROL* (line 12). Specify Solver routine WAALS (line 13). *Indirectly* specify the variable to be solved (line 14). View the solution and return to label GAS2 (lines 15 – 16).

```
17 LBL "WAALS"
18 RCL "P"
19 RCL "n"
20 X↑2
21 RCL× "a"
22 RCL "V"
23 X↑2
24 ÷
25 +
26 RCL "V"
27 RCL "n"
28 RCL× "b"
29 -
30 ×
```

Lines 17 – 34, the Solver routine WAALS: Express the van der Waals equation such that its right side equals 0.

```
31 0.082057
32 RCL× "n"
33 RCL× "T"
34 -
35 END
```

Part 2. Use the van der Waals equation of state to calculate the pressure exerted by 0.250 mole of carbon dioxide in 0.275 liter at 373 K, and compare this value with the value expected for an ideal gas. For CO_2, $a = 3.59$ liters^2 – atmosphere/mole^2, and $b = 0.0427$ liter/mole.

Execute GAS2.

[XEQ] GAS2

```
x: 0.0000
 P | V | N | T | A | B
```

Enter the values for the known variables.

.250 N
.275 V
373 T
3.59 A
.0427 B

```
b=0.0427
 P | V | N | T | A | B
```

Enter guesses of 10 and 30 for P, and solve for P.

10 P
30 P
P

```
P=25.9816
 P | V | N | T | A | B
```

Using the van der Waals equation of state, the predicted pressure is 25.9816 atmospheres.

Now use the ideal gas equation to predict the pressure. Simply supply the value 0 for a and b and solve for P. The previously calculated value for P serves as an initial guess.

0 A
0 B
P

```
P=27.8248
 P | V | N | T | A | B
```

The ideal gas equation predicts a pressure of 27.8248 atmospheres. (The actual observed pressure is 26.1 atmospheres.)

Exit from GAS2.

[EXIT]

```
Y: 27.8248
X: 27.8248
```

More on How the Solver Works

The Root(s) of a Function

To use the Solver, you have learned that you first create a program that expresses the equation such that its right side equals 0 (by subtracting the terms on the right side from both sides of the equation). If the equation has more than one variable, you must, after selecting the Solver application, supply values for all but the one unknown variable. At this point, your equation has taken the form $f(x) = 0$, where x is the unknown variable, and $f(x)$ is a mathematical shorthand for the *function* that defines x. Consider the equation

$$2x^2 + xy + 10 = 3xz + 2yz$$

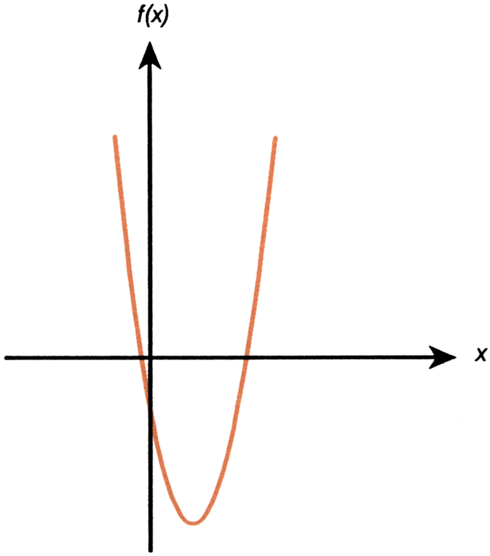
Setting the equation equal to 0 by subtracting the terms on the right side from both sides gives

$$2x^2 + xy + 10 - 3xz - 2yz = 0$$

To use the Solver, you now write a program that declares the variables x, y, and z and expresses the equation. When you select the Solver application and, for example, supply the value 2 for y, and 3 for z, by substitution the equation becomes

$$2x^2 - 7x - 2 = 0$$

where x is the unknown variable and $f(x) = 2x^2 - 7x - 2$. Each value x for which $f(x) = 0$ is called a *root* of the function. The Solver iteratively seeks a root for $f(x)$ by evaluating the function repeatedly at estimates of x, and comparing the results to previous estimates. Using a complex algorithm, the Solver intelligently "predicts" a new estimate of where the graph of $f(x)$ might cross the x-axis. Here is a graph of the function $f(x) = 2x^2 - 7x - 2$. The graph shows two roots. (The example on pages 110–112 calculates these roots.)



All except one of the functions in the examples in this section are functions of one variable x only. Remember, though, that the situations described in the examples apply equally to multivariable functions, since multivariable functions *become* single variable functions when, in the Solver application, you supply values for the known variables.

The Solver's Ability to Find a Root

For the Solver to find a root, the root has to exist within the range of numbers of the calculator, and the function must be mathematically defined where the iterative search occurs. The Solver always finds a root if one or more of the following conditions is met:

- Two estimates yield $f(x)$ values with opposite signs, and the function's graph crosses the x-axis in at least one place between those estimates (figure 3-1a).
- $f(x)$ always increases or always decreases as x increases (figure 3-1b).
- The graph of $f(x)$ is either concave everywhere or convex everywhere (figure 3-1c).
- If $f(x)$ has one or more local minima or maxima, each occurs singly between adjacent roots of $f(x)$ (figure 3-1d).

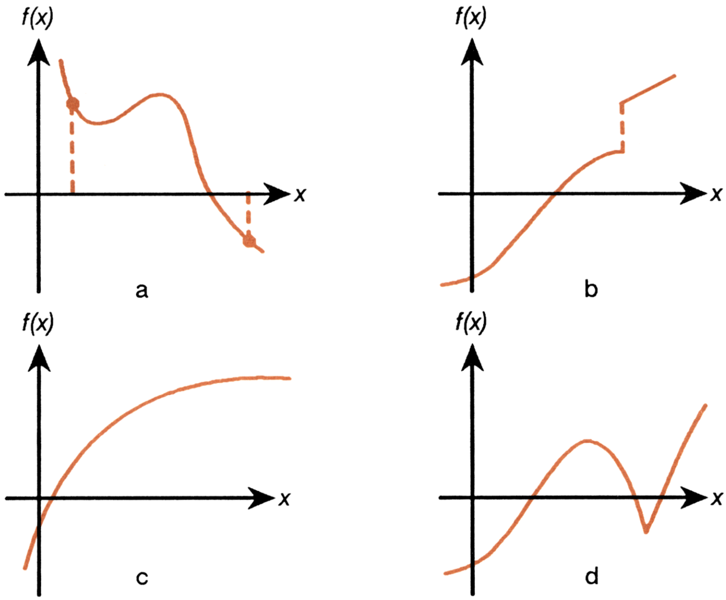


Figure 3-1. Functions for Which a Root Can Be Found

In most situations, the calculated root is an accurate estimate of the theoretical, infinitely precise root of the function. An *ideal* solution is one for which $f(x)$ exactly equals 0. However, a nonzero value for $f(x)$ is often also acceptable, because it results from approximating the root with limited (12-digit) precision.

Interpreting the Results of the Solver

The Solver returns data to the stack registers on completion of its iterative search for a root of the specified function, and in four conditions, returns a message to the display. These messages and data can help you interpret the results of the search:

- The X-register contains the best guess. This guess *may or may not be a root* of the function.
- The Y-register contains the previous guess.
- The Z-register contains the value of the function $f(x)$ evaluated at the best guess.
- The T-register contains a code 0–4 that indicates the Solver's interpretation of its search for a root. (This code is displayed in the current display mode; in FIX 4, code 0 is displayed as `0.0000`.).

Code in T-register	Interpretation	Message
0	The Solver has found a root.	
1	The Solver has generated a sign reversal in *f(x)* at neighboring values of *x*, but *f(x)* has been strongly diverging from 0 as *x* approaches the two neighbors from both sides.	`Sign Reversal`
2	The Solver has found an approximation to a local minimum or maximum of the numerical absolute value. If the solution is $\pm 9.99999999999 \times 10^{499}$, it corresponds to an asymptotic extremum.	`Extremum`
3	One or both initial guesses lie outside the domain of *f(x)*. That is, *f(x)* returns an error when evaluated at the guess points.	`Bad Guess(es)`
4	*f(x)* returns the same value at every point evaluated by the Solver.	`Constant?`

When a Root Is Found. There are two cases in which a root is found:

- In case 1, the calculated root sets $f(x)$ exactly equal to 0 (figure 3-2a).
- In case 2, the calculated root does not set $f(x)$ exactly equal to 0, but is a 12-digit number adjacent to the place where the function's graph crosses the x-axis (figure 3-2b). This occurs when the final two estimates are *neighbors* (they differ by 1 in the 12th digit) and $f(x)$ is positive for one estimate and negative for the other. In most cases, $f(x)$ will be relatively close to 0.

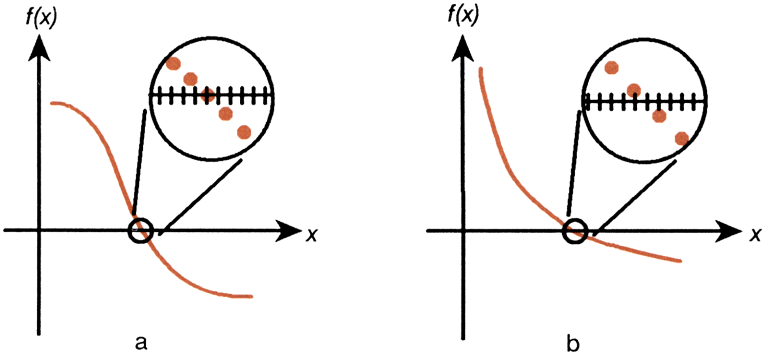


Figure 3-2. Case When A Root Is Found

In both cases, the code in the T-register is a 0 and no message is displayed. You can differentiate between the two cases by:

- Viewing the contents of the Z-register (the value of $f(x)$ at the calculated root). For a case 2 solution, it will be a nonzero number.
- Comparing the best guess (the contents of the X-register) and the previous guess (the contents of the Y-register). For a case 2 solution, the guesses differ by 1 in the 12th digit.
- Immediately solving again for the variable. For a case 2 solution, the Solver will return the message `Sign Reversal` on the second attempt to find the root.

Example: A Case 1 Solution with Two Roots. Find the two roots of the equation

$$2x^2 - 7x - 2 = 0$$

Express the function in program AA.

```
00 { 25-Byte Prgm }
01 LBL "AA"
02 MVAR "X"
03 RCL "X"
```

```
04 X↑2
05 2
06 ×
07 7
08 RCL× "X"
09 -
10 2
01 -
12 END
```

Set the display format to ALL. Select the Solver application and then program AA.

■ [DISP] ALL	x: 0
■ [SOLVER]	X
AA	

Enter guesses of 1 and 5 for x. Solve for x.

1 X	X=3.76556443708
5 X	X
X	

Roll the stack contents down to see the previous guess.

[R↓]	x: 3.76556443708
	X

The estimates are the same in all 11 decimal places. Roll the stack contents down to see the value of $f(x)$ at the root.

[R↓]	x: 0
	X

$f(x)$ is exactly 0. Now enter guesses of -0.1 and -1 for the second root and solve.

.1 [+/-] X	X=-2.65564437075E-1
1 [+/-] X	X
X	

Roll the stack contents down to see the value of $f(x)$ at the root. Again, $f(x)$ is exactly 0.

[R↓] [R↓]

```
x: 0
 X
```

Exit from the Solver and return the display format to FIX 4.

[EXIT] [EXIT]
■[DISP] FIX 4 [ENTER]

```
y: 0.0000
x: 0.0000
```

Example: A Case 2 Solution. In the example on pages 101 – 105 in this chapter, you found the value of the pressure P in the ideal gas equation of state given values for the other variables V, n, and T.

Using the same values for the variables V, n, and T, solve again for P.

Set the display format to ALL.

■[DISP] ALL

```
Select Solve Program
 GAS2
```

Start program GAS2. (Reenter the program if you have cleared it from the calculator.)

[XEQ] GAS2

```
x: 0
 P  V  N  T  A  B
```

Enter the values for the known variables and solve for the pressure.

.25 N
.275 V
373 T
0 A
0 B
P

```
P=27.8247827273
 P  V  N  T  A  B
```

Roll the stack down to see the previous estimate.

[R↓]

```
x: 27.8247827272
 P  V  N  T  A  B
```

The estimates differ by 1 in the last decimal place. Roll the stack down to see the value of $f(x)$.

[R↓]

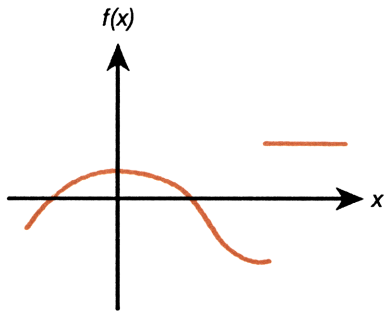
```
x: 0.00000000001
 P | V | N | T | A | B
```

The value of $f(x)$ at the root is a very small nonzero number. The root is not an exact root, but it is a very good approximation. Exit from the program and return the display format to FIX 4.

[EXIT]
■ [DISP] FIX 4 [ENTER]

```
Y: 0.0000
X: 1.0000E-11
```

Problems That Require Special Consideration. Some types of problems require special consideration. The following function has a discontinuity that crosses the x-axis.



The Solver will return an x-value adjacent to the discontinuity. The value of $f(x)$ may be relatively large.

Example: A Discontinuous Function. Find the root of the equation

$$IP(x) - 1.5 = 0$$

Express the function in program BB.

```
00 { 18-Byte Prgm }
01 LBL "BB"
02 MVAR "X"
03 RCL "X"
04 IP
05 1.5
06 -
07 END
```

Select the Solver, select program BB, provide guesses of 0 and 5, and solve for x.

■ [SOLVER] BB
0 X
5 X
X

```
X=2.0000
 X
```

The Solver finds a root at $x = 2.0000$. Now check the value of $f(x)$.

[R↓] [R↓]

```
x: -0.5000
 X
```

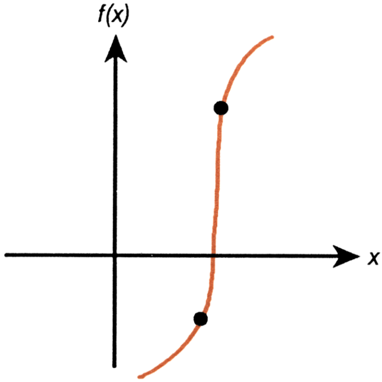
The value of $f(x)$ seems relatively large. This indicates that you should further evaluate the function. By plotting the function, you find that the root at $x = 2.0000$ is in fact a discontinuity, and not a true zero crossing.

Exit from the Solver.

[EXIT] [EXIT]

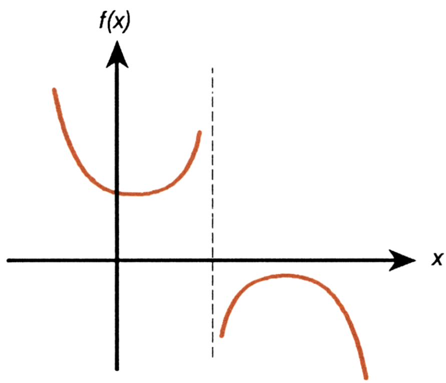
```
y: 0.0000
x: -0.5000
```

Finally, consider the following function. This function has a very steep slope in the area of the root. Evaluation of the function at either neighbor may return a very large value even though the function has a true root between the neighbors.



Use care in interpreting the results of the Solver. The Solver is most effective when used in conjunction with your own analysis of the function you are evaluating.

A Sign Reversal. The values of the following function are approaching infinity at the location x_0 where the graph changes sign.



The function has a *pole* at x_0. When the Solver evaluates such a function, it returns the message `Sign Reversal.`

Example: A Pole. Find the root of the equation

$$\frac{x}{(x^2 - 6)} - 1 = 0$$

As x approaches $\sqrt{6}$, $f(x)$ becomes a very large positive or negative number.

Express the function in program CC.

```
00 { 23-Byte Prgm }
01 LBL "CC"
02 MVAR "X"
03 RCL "X"
04 RCL "X"
05 X↑2
06 6
07 -
08 ÷
09 1
10 -
11 END
```

Select the Solver and then select program CC.

■ [SOLVER] [CC]

```
x: 0.0000
X
```

Provide guesses of 2.3 and 2.7, and solve for x.

2.3 [X]
2.7 [X]
[X]

```
X=2.4495
Sign Reversal
```

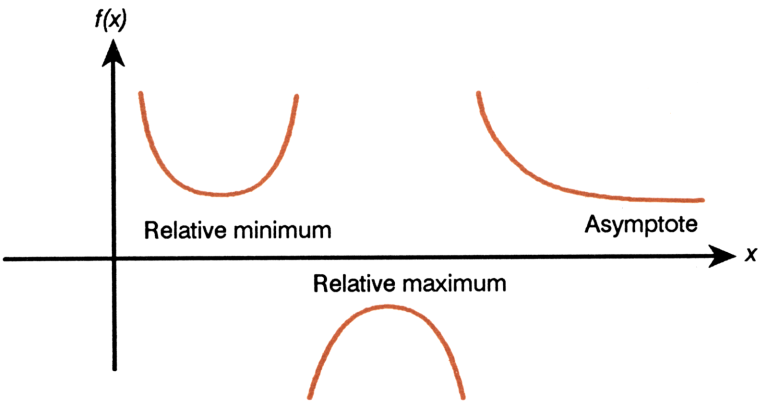
The initial guesses yielded opposite signs for $f(x)$. The interval between successive estimates was then narrowed until two neighbors were found. These neighbors made $f(x)$ approach a pole instead of the x-axis. The function *does* have roots at -2 and 3, which can be found by entering better guesses.

Exit from the Solver.

[EXIT] [EXIT]

```
Y: 2.4495
X: 2.4495
```

An Extremum. When the Solver returns the message `Extremum`, it has found an approximation to a local minimum or maximum of the numerical absolute value of the function. If the solution (the value in the X-register) is $+/-$ $9.99999999999 \times 10^{499}$, the Solver has found an *asymptotic* extremum.



Example: A Relative Minimum. Find the solution of the parabolic equation

$$x^2 - 6x + 13 = 0$$

(It has a minimum at $x = 3$.)

Express the function in program DD.

```
00 { 23-Byte Prgm }
01 LBL "DD"
02 MVAR "X"
03 RCL "X"
04 X↑2
05 6
06 RCL× "X"
07 -
08 13
09 +
10 END
```

Select the Solver application and then program DD.

■ [SOLVER] DD

```
x: 0.0000
 X
```

Provide guesses of 0 and 10 and solve for x.

0 X
10 X
X

```
X=3.0000
Extremum
```

Exit from the Solver.

[EXIT] [EXIT]

```
y: 3.0000
x: 3.0000
```

Example: An Asymptote. Find the solutions for the equation

$$10 - \frac{1}{x} = 0$$

Express the function in program EE.

```
00 { 17-Byte Prgm }
01 LBL "EE"
02 MVAR "X"
03 10
04 RCL "X"
05 1/X
06 -
07 END
```

Select the Solver application and then program EE.

■ [SOLVER] EE

```
x: 0.0000
 X
```

Enter guesses of 0.005 and 5, and solve for x.

.005 X
5 X
X

```
X=0.1000
 X
```

The Solver finds a root at $x = 0.1000$. Now enter guesses that have negative values.

1 [+/-] X
2 [+/-] X
X

```
X=-1.0000E500
Extremum
```

The Solver finds an asymptotic extremum. (Press ■ [SHOW] to verify that the solution is actually $-9.99999999999 \times 10^{499}$.) It's apparent from inspecting the equation that if x is a negative number, the smallest that $f(x)$ can be is 10; $f(x)$ *approaches* 10 as x becomes a large negative number.

Exit from the Solver.

[EXIT] [EXIT]

```
Y: -5.9246E498
X: -1.0000E500
```

Bad Guess(es). The Solver returns the message `Bad Guess(es)` when one or both initial guesses lie outside the domain of the function. (If a guess lies outside the domain of the function, the function returns a math error when evaluated at that guess point.)

Example: A Math Error. Find the root of the equation

$$\sqrt{\frac{x}{(x + 0.3)}} - 0.5 = 0$$

Express the function in program FF.

```
00 { 26-Byte Prgm }
01 LBL "FF"
02 MVAR "X"
03 RCL "X"
04 0.3
05 RCL+ "X"
06 ÷
07 SQRT
08 0.5
09 -
10 END
```

Select the Solver application and then program FF.

■ [SOLVER] FF

```
x: 0.0000
X
```

First attempt to find a positive root, using guesses 0 and 10.

0 X
10 X
X

```
X=0.1000
X
```

The Solver finds a root at $x = 0.1$. Now attempt to find a negative root using guesses of -0.1 and -0.2. Note that the function is undefined for values of x between 0 and -0.3, since those values produce a positive denominator but a negative numerator, causing a negative square root. Although the HP-42S can execute arithmetic operations with complex numbers, the Solver cannot find a complex number solution. If evaluation of $f(x)$ returns a complex number, the Solver considers the function undefined at that x-value.

.1 [+/-] [X]
.2 [+/-] [X]
[X]

```
X=-0.2000
Bad Guess(es)
```

Exit from the Solver.

[EXIT] [EXIT]

```
Y: -0.1000
X: -0.2000
```

A Constant. The Solver returns the message `Constant?` when it finds that $f(x)$ returns the same value at every sample point x. Such a situation can occur if guesses are confined to a local "flat" region of a function.

Example: A Local Flat Region. Find the root of the equation

$$\frac{1}{x} - 10 = 0$$

Express the function in program GG.

```
00 { 17-Byte Prgm }
01 LBL "GG"
02 MVAR "X"
03 RCL "X"
04 1/X
05 10
06 -
07 END
```

Select the Solver and then program GG.

■ SOLVER GG

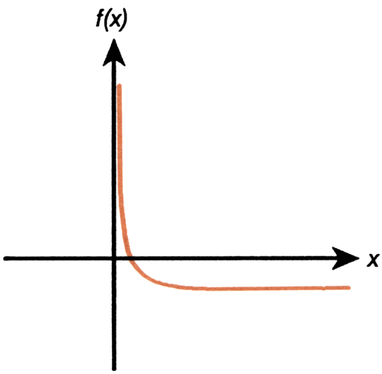
```
x: 0.0000
 X
```

Supply guesses of 10^{20} and 10^{30}.

E 20 X
E 30 X
X

```
X=1.0000E500
Constant?
```

In this region of the function, the value of $f(x)$ is, within the 12-digit precision of the calculator, the same at every sample point. Here is a graph of the function.



Try guesses of 0 and 10.

0 X
10 X
X

```
X=0.1000
 X
```

The Solver finds the root at $x = 0.1$. Exit from the Solver.

EXIT EXIT

```
Y: 0.1000
x: 0.1000
```

Round-Off Error and Underflow

Round-Off Error. The 12-digit precision of the calculator is adequate for almost all cases. However, round-off errors can sometimes affect Solver results. For example,

$$[(|x| + 1) + 10^{15}]^2 - 10^{30} = 0$$

has no roots because $f(x)$ is always positive. However, given initial guesses of 1 and 2, the Solver returns the answer 1.0000 because of round-off error.

Round-off error can also cause the Solver to fail to find a root. The equation

$$|x^2 - 7| = 0$$

has a root at $\sqrt{7}$. However, no 12-digit number *exactly* equals $\sqrt{7}$, so the calculator can never make the function equal to 0. Furthermore, the function never changes sign. The Solver returns the message `Extremum`. However, the final estimate of x is the best possible 12-digit approximation of the root when the routine ends.

Underflow. *Underflow* can occur when the magnitude of a number is smaller than the calculator can represent; in such a case, it will substitute the number 0. This can affect the Solver's results. For example, consider the equation

$$\frac{1}{x^2} = 0$$

whose root is infinity. Because of underflow, the Solver returns a very large (finite) value as a root. (The calculator cannot represent infinity, anyway.)

4

Integration

In this chapter, the following topics are covered:

- Basic use of the Integration application.
- Approximating an integral that has an infinite upper or lower limit.
- Using Integration and the Solver interactively.
- More on how Integration works.

Basic Integration

The procedure for execution of the Integration application is:

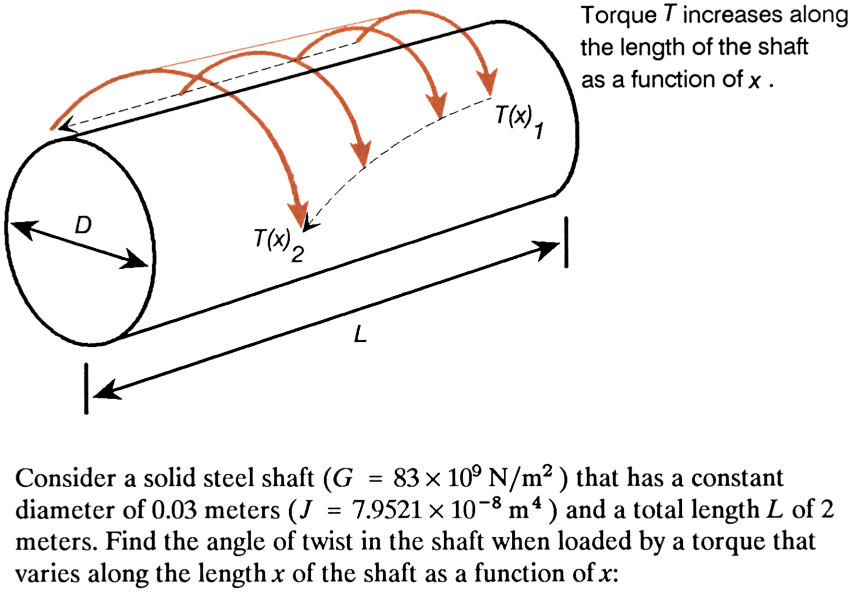
1. Create a program that:
   a. Uses MVAR to define the variable(s) in the integrand (the function to be integrated).
   b. Expresses the integrand. (Note that each variable in the integrand must be recalled to the X-register.)
2. Apply the Integration application to the program.
   a. Select the Integration application (press ■ [∫f(x)]).
   b. Select the program by pressing the corresponding menu key.
   c. Specify the values for any known variables in the integrand. Select the variable of integration.
   d. Specify the values for *LLIM*, *ULIM*, and *ACC*.
   e. Press [∫] to begin the calculation.

Example: Basic Integration. The angle of twist in a round shaft under torsional loading is calculated by evaluating the following integral.

$$\theta = \int_0^L \frac{T}{JG}\, dx$$

where:

θ is the angle of twist of the shaft (in radians).
L is the length of the shaft (in meters).
T is the torque applied to the shaft (in Newton-meters).
J is the polar moment of inertia of the shaft (in meters^4).
G is the shear modulus of the shaft material (in Newtons/meters^2).



Consider a solid steel shaft ($G = 83 \times 10^9$ N/m^2) that has a constant diameter of 0.03 meters ($J = 7.9521 \times 10^{-8}$ m^4) and a total length L of 2 meters. Find the angle of twist in the shaft when loaded by a torque that varies along the length x of the shaft as a function of x:

$$T = 13x^4 + 8x^3 + 15x^2 + 9x + 6$$

For programming purposes, use Horner's method to expand the polynomial.

$$T = (((13x + 8)x + 15)x + 9)x + 6$$

Substituting this expression for T, the equation becomes

$$\theta = \int_0^L \frac{(((13x+8)x+15)x+9)x+6}{JG}\,dx$$

Express the integrand in the program TORQUE.

Program:	**Comments:**
`00 { 53-Byte Prgm }`	
`01 LBL "TORQUE"`	
`02 MVAR "X"`	Lines 02 – 04: Declare the variables.
`03 MVAR "J"`	
`04 MVAR "G"`	
`05 13`	Lines 05 – 19: Express the integrand.
`06 RCL× "X"`	
`07 8`	
`08 +`	
`09 RCL× "X"`	
`10 15`	
`11 +`	
`12 RCL× "X"`	
`13 9`	
`14 +`	
`15 RCL× "X"`	
`16 6`	
`17 +`	
`18 RCL÷ "J"`	
`19 RCL÷ "G"`	
`20 END`	

Select the Integration application.

■ [∫f(x)]	`Select ∫f(x) Program` `TORQ`

Select program TORQUE.

TORQ	`Set Vars; Select ∫var` `X  J  G`

Supply the known values for J and G, and specify the variable of integration X.

7.9521 [E] 8 [+/-] J
83 [E] 9 G
X

```
x: 83,000,000,000.0
LLIM ULIM ACC ∫
```

Specify the lower limit (0), the upper limit L (2), and an accuracy factor of 0.01.

0 LLIM
2 ULIM
.01 ACC

```
ACC=0.0100
LLIM ULIM ACC ∫
```

Start the calculation.

∫

```
∫=0.0281
LLIM ULIM ACC ∫
```

The shaft twists through an angle θ = 0.0281 radians (1.6077 degrees). Exit from the Integration application.

[EXIT] [EXIT] [EXIT]

```
y: 0.0003
x: 0.0281
```

Approximating an Integral That Has an Infinite Limit

It is often of interest to evaluate an *improper* integral (an integral that has an infinite upper or lower limit). An improper integral with an infinite upper limit

$$\int_0^\infty f(x)\,dx$$

is calculated "by hand" by evaluating the equivalent expression

$$\lim_{a\to\infty} \int_0^a f(x)\,dx$$

You cannot use the HP-42S to directly evaluate such an expression. You can, however, *approximate* an answer by substituting a large number for the infinite limit.

Example: Evaluating an Integral That Has an Infinite Upper Limit. Calculate the integral

$$\int_0^\infty \frac{dx}{1 + x^2}$$

by hand. Then approximate the integral with the HP-42S.

Part 1. The result is calculated by hand as follows.

$$\begin{aligned}\int_0^\infty \frac{dx}{1 + x^2} &= \lim_{a \to \infty} \int_0^a \frac{dx}{1 + x^2} \\ &= \lim_{a \to \infty} (\arctan a) \\ &= \frac{\pi}{2}\end{aligned}$$

Use the HP-42S to calculate $\pi/2$ to 12-digit precision.

■ DISP ALL	Y: 0
■ π 2 ÷	X: 1.5707963268

Part 2. Use the Integration application to evaluate the same integral, using the value 1,000 to approximate the upper limit. First, express the integrand in the program INFIN.

```
00 { 20-Byte Prgm }
01 LBL "INFIN"
02 MVAR "X"
03 RCL "X"
04 X↑2
05 1
06 +
07 1/X
08 END
```

Select the Integration application and then program INFIN.

■ [∫f(x)] INFIN

```
Set Vars; Select ∫var
 X
```

Select the variable of integration.

X

```
x: 1.5707963268
LLIM ULIM ACC ∫
```

Specify the lower limit (0), the upper limit approximation (1,000), and an accuracy factor of 0.01.

0 LLIM
1000 ULIM
.01 ACC

```
ACC=0.01
LLIM ULIM ACC ∫
```

Calculate the integral.

∫

```
∫=1.57020935993
LLIM ULIM ACC ∫
```

Using an upper limit of 1,000, and an accuracy factor of 0.01, the calculator returns the result 1.57020935993. The calculation takes about 36 seconds and is correct to three decimal places.

Exit from the Integration application and return the display format to FIX 4.

[EXIT] [EXIT] [EXIT]
■ [DISP] FIX 4 [ENTER]

```
Y: 0.0156
x: 1.5702
```

The following table summarizes results and calculation times for upper limit approximations of 100, 1,000, and 10,000, and accuracy factors of 0.01 and 0.0001.

Acc. Factor	ULIM	Result	Calc. Time (seconds)
		($\frac{\pi}{2}$ actual) 1.5707963268	
0.01	100	1.57518831857	5
	1,000	1.57020935993	36
	10,000	1.57088603739	140
0.0001	100	1.5607891695	18
	1,000	1.56979476064	69
	10,000	1.57069673168	279

Note that the principle determining factor in the accuracy of the result is the value of the upper-limit approximation, not the accuracy factor. Also note that the calculations using an accuracy factor of 0.0001 require about twice the time of those using an accuracy factor of 0.01.

In general, when you are approximating an integral, assess the extent to which you are constraining the accuracy of the true integral with the approximation of the limit, and choose an accuracy factor wisely. If the limit that you substitute results in only a rough approximation of the true integral, it makes little sense to calculate the approximation to a high degree of accuracy.

Using the Solver and Integration Interactively

In the first example in this chapter, you found the twist angle θ at the end of a shaft by integrating the applied torque with respect to x. (The torque varied as a function of the position x along the shaft.) You were limited, in that example, to solving specifically for the twist angle θ. In general, for the equation

$$I = \int_{LLIM}^{ULIM} f(x)\,dx \text{ (calculated to accuracy } ACC\text{)}$$

the Integration application enables you to solve *only* for the value I of the integral. To solve for I, you:

- Write a program P that defines the integrand $f(x)$.
- Specify values for the known variables in the integrand.
- Specify the variable of integration.
- Specify values for the variables *LLIM*, *ULIM*, and *ACC*.

However, by writing a program S for the Solver that declares each variable in the equation and *invokes the Integration application on program P*, you can solve for any of the variables in the equation:

- I
- The variables in the integrand $f(x)$.
- *LLIM*, *ULIM*.

In the following example, you'll solve for the length L of a shaft (the variable *ULIM* in the Integration application) in the angle-of-twist equation.

Example. Using the Solver and Integration Interactively. Restating the equation for twist in a shaft under torsional loading:

$$\theta = \int_0^L \frac{T}{JG}\,dx$$

Consider again the solid steel shaft of the first example in this chapter. For this shaft, $G = 83 \times 10^9 \ \mathrm{N/m^2}$ and $J = 7.9521 \times 10^{-8} \ \mathrm{m^4}$. The shaft is subjected to the same torsional loading T as in the first example. That loading varies along the length x of the shaft as a function of x.

$$T = 13x^4 + 8x^3 + 15x^2 + 9x + 6.$$

Find the length L that results in a twist angle θ of 0.1396 radians (8 degrees).

The variables in the equation are θ, L, T, J, and G. The unknown variable L is the upper limit of integration *ULIM*.

Part 1. Write a Solver program SHAFT that:

- Declares each variable in the equation.
- Expresses the equation such that its right side equals 0.

$$\int_0^L \frac{T}{JG}\, dx - \theta = 0$$

Program: | **Comments:**

```
00 { 60-Byte Prgm }
01 LBL "SHAFT"
```

```
02 MVAR "THETA"
03 MVAR "G"
04 MVAR "J"
05 MVAR "LLIM"
06 MVAR "ULIM"
07 MVAR "ACC"
08 MVAR "X"
```

Lines 02–08: Declare the variables in the equation.

```
09 PGMINT "TORQUE"
10 INTEG "X"
11 RCL- "THETA"
```

Lines 09–11: Express the equation such that its right side equals 0. First, calculate the first term of the equation (the integral) (lines 09–10). The value of the integral is returned to the X-register. Subtract the second term (*THETA*) (line 11).

```
12 END
```

In lines 09 – 10, the integral is calculated using the current value of *ULIM*, which is iteratively supplied by the Solver as it searches for a solution. Note that the specified integration program is TORQUE from the first example in the chapter. If you've deleted this program, you need to key it into the calculator now.

Part 2. Select the Solver application and then program SHAFT.

■ [SOLVER] SHAFT

```
x: 0.0000
THETA  G  J  LLIM ULIM ACC
```

(The variable *X* is on the second line of the menu.) Enter values for the known variables.

.1396 THETA
83 [E] 9 G
7.9521 [E] 8 [+/-] J
0 LLIM
.01 ACC

```
ACC=0.0100
THETA  G  J  LLIM ULIM ACC
```

Now solve for the upper limit *L*, providing initial guesses of 1 and 10.

1 ULIM
10 ULIM
ULIM

```
ULIM=2.9528
THETA  G  J  LLIM ULIM ACC
```

The shaft must be 2.9528 meters long to twist through an angle of 0.1396 radians.

Exit from the Solver application.

[EXIT] [EXIT]

```
Y: 2.9528
x: 2.9528
```

More on How Integration Works

The Accuracy Factor and the Uncertainty of Integration

The Integration algorithm calculates the integral of a function $f(x)$ by computing a weighted average of the function's values at many values of x (sample points) within the interval of integration. The accuracy of the result depends on the number of sample points considered; generally, the more the sample points, the greater the accuracy. There are two reasons why you might want to *limit* the accuracy of the integral:

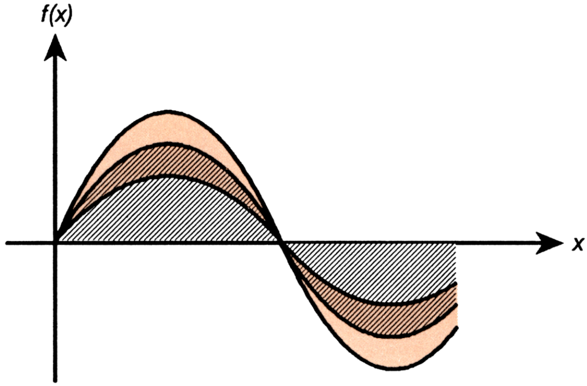
1. The length of time to calculate the integral increases as the number of sample points increases.
2. There are inherent inaccuracies in each calculated value of $f(x)$:
   a. *Empirically-derived* constants in $f(x)$ may be inaccurate. If, for example, $f(x)$ contains empirically-derived constants that are accurate to only two decimal places, it is of little value to calculate the integral to the full (12-digit) precision of the calculator.
   b. If $f(x)$ models a physical system, there may be inaccuracies in the model.
   c. The calculator itself introduces round-off error into each computation of $f(x)$.

To indirectly limit the accuracy of the *integral*, specify the *accuracy factor* of the *function*, defined as

$$ACC = \left| \frac{\text{true value of } f(x) - \text{computed value of } f(x)}{\text{computed value of } f(x)} \right|$$

The accuracy factor is your estimation of the (decimal form of the) percent error in each computed value of $f(x)$. This value is stored in *ACC*. The accuracy factor is related to the *uncertainty of integration* (a measurement of the accuracy of the integral) by:

$$\textit{uncertainty of integration} = \textit{accuracy factor} \times \int |f(x)|\, dx$$



The striped area is the value of the integral. The orange-shaded area is the value of the uncertainty of integration. It is the weighted sum of the errors of each computation of $f(x)$. You can see that at any point x, the uncertainty of integration is proportional to $f(x)$.

The Integration algorithm uses an iterative method, doubling the number of sample points in each successive iteration. At the end of each iteration, it calculates both the integral and the uncertainty of integration. It then compares the value of the integral calculated during that iteration with the values calculated during the two previous iterations. If the difference between any one of these three values and the other two is less than the uncertainty of integration, the algorithm stops. The current value of the integral is returned to the X-register, and the uncertainty of integration is returned to the Y-register.

It is extremely unlikely that the errors in each of the three successive calculations of the integral – that is, the differences between the actual integral and the calculated values – would all be larger than the disparity among the approximations themselves. Consequently, the error in the final calculated value will almost certainly be less than the uncertainty of integration.

Example: The Accuracy Factor and the Uncertainty of Integration. Certain problems in communications theory (for example, pulse transmissions through idealized networks) require calculating an integral (sometimes called the *sine integral*) of the form

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

Find Si (2).

First, write a program that expresses the function.

```
00 { 16-Byte Prgm }
01 LBL "SI"
02 MVAR "X"
03 RCL "X"
04 SIN
05 RCL÷ "X"
06 END
```

Set the display format to ALL. Set the angular mode to RAD.

■ DISP ALL
■ MODES RAD

```
Y: 0
X: 0
```

Select the Integration application and then program SI.

■ ∫f(x) SI

```
Set Vars; Select ∫var
X
```

Select the variable of integration *X*, then enter a lower limit of 0 and an upper limit of 2.

X
0 LLIM
2 ULIM

```
ULIM=2
LLIM ULIM ACC ∫
```

Since the function

$$f(x) = \frac{\sin x}{x}$$

is a purely mathematical expression containing no empirically-derived constants, the only constraint on the accuracy of the function is the round-off error introduced by the calculator. It is, therefore, at least analytically reasonable to specify an accuracy factor of 0.00000000001 (1×10^{-11}).

E 11 +/- ACC

```
ACC=0.00000000001
LLIM ULIM ACC        ∫
```

Calculate the integral.

∫

```
∫=1.6054129768
LLIM ULIM ACC        ∫
```

Check the uncertainty of integration.

x≷y

```
x: 2.10542218026E-11
LLIM ULIM ACC        ∫
```

The uncertainty of integration is significant only with respect to the last digit of the integral. The calculation took about 19 seconds. If you can accept a less accurate answer, you can shorten the calculation time. Try an accuracy factor of 0.001.

.001 ACC ∫

```
∫=1.60541531589
LLIM ULIM ACC        ∫
```

Check the uncertainty of integration.

x≷y

```
x: 1.60600822892E-3
LLIM ULIM ACC        ∫
```

The error of integration is much larger now. However, it is still relatively small compared to the value of the integral, and the calculation takes only 3 seconds.

Exit from the Integration application and return the display format to FIX 4.

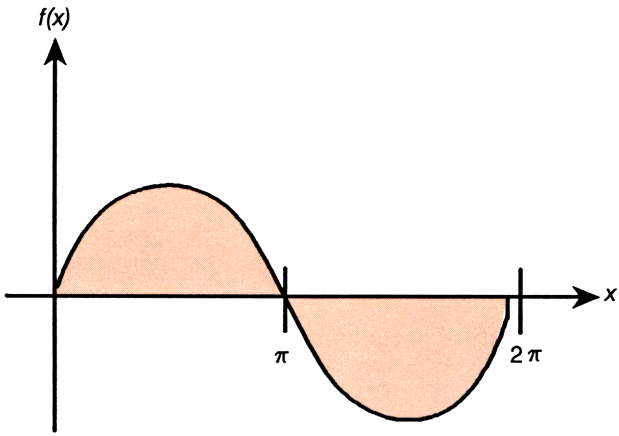
[EXIT] [EXIT] [EXIT]
■ [DISP] FIX 4 [ENTER]

```
Y: 1.6054
X: 0.0016
```

Example: A Problem Where the Uncertainty of Integration Is Relatively Large. In the previous example, the uncertainty of integration was relatively small compared to the value of the integral. This is because the value of the function was always positive within the interval of integration. Now consider the simple function

$$f(x) = \sin x$$

Integrate the function from $x = 0$ to $x = 6$ (radians).



By inspection, you can see that the value of the integral is a small positive number, since the area with positive value from 0 to π is almost cancelled by the area with negative value from π to 6.

Write the program that expresses the function.

```
00 { 14-Byte Prgm }
01 LBL "SIN"
02 MVAR "X"
03 RCL "X"
04 SIN
05 END
```

Set the angular mode to RAD. Select the Integration application and then program SIN.

■ [MODES] RAD
■ [∫f(x)]
SIN

```
Set Vars; Select ∫var
 X
```

Select the variable of integration *X*, enter the lower and upper limits (0 and 6), and an accuracy factor of 0.01. Then integrate with respect to *x*.

X
0 LLIM
6 ULIM
.01 ACC
∫

```
∫=0.0398
LLIM ULIM ACC ∫
```

Now check the uncertainty of integration.

[x≷y]

```
x: 0.0398
LLIM ULIM ACC ∫
```

The uncertainty of integration is large compared to the value of the integral.

Exit from the Integration application.

[EXIT] [EXIT] [EXIT]

```
y: 0.0398
x: 0.0398
```

Conditions That Can Cause Incorrect Results

Although the integration algorithm in the HP-42S is one of the best available, in certain situations it – like all algorithms for numeric integration – might give you an incorrect answer. *The possibility of this occurring is very remote.* The integration algorithm has been designed to give accurate results for almost any smooth function. Only for functions that exhibit *extremely* erratic behavior is there any substantial risk of obtaining an inaccurate answer. Such functions rarely occur in problems related to actual physical systems.

Example: A Condition That Causes an Incorrect Result. Consider the approximation of

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

Since you're evaluating this integral numerically, you might think that you should represent the upper limit of integration with a large number, say 100,000. Try it and see what happens. First write a program that expresses $f(x)$.

```
00 { 17-Byte Prgm }
01 LBL "XEX"
02 MVAR "X"
03 RCL "X"
04 ENTER
05 +/-
06 E↑X
07 ×
08 END
```

Now select the Integration application and then program XEX.

■ [∫f(x)] [XEX]

```
Set Vars; Select ∫var
 X
```

Select the variable of integration X, then enter the lower and upper limits and an accuracy factor of 0.001.

X
0 LLIM
E 5 ULIM
.001 ACC

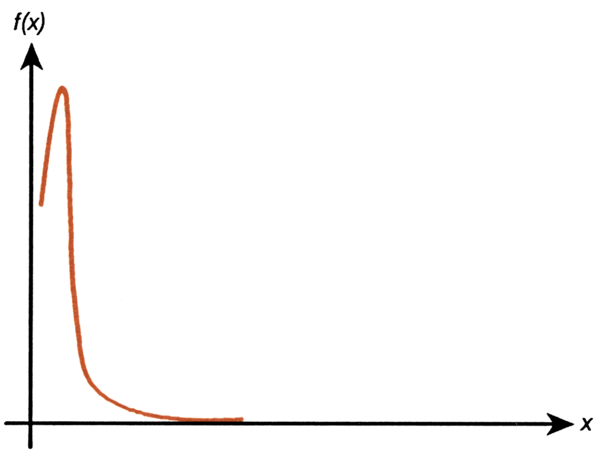
```
ACC=0.0010
LLIM ULIM ACC       ∫
```

Integrate with respect to x. (Stay in the Integration application after executing this calculation. You will integrate this function again in the next section.)

∫

```
∫=0.0000
LLIM ULIM ACC       ∫
```

The answer is clearly incorrect, since the actual integral of $f(x) = xe^{-x}$, evaluated from 0 to ∞, is exactly 1. But the problem is *not* that you represented ∞ by 100,000, since the actual integral of this function from 0 to 100,000 is very close to 1. The reason you obtained an incorrect answer becomes apparent if you look at the graph of $f(x)$ over the interval of integration.



The graph has a spike (illustrated here with a greatly exaggerated width) very close to the origin . Because no sample point discovered the spike, the algorithm assumed that $f(x)$ was equal to 0 throughout the interval of integration. Even if you increased the number of sample points by specifying an accuracy factor of 1×10^{-11}, none of the additional sample points would discover the spike when this particular function is integrated over this particular interval.

Subdividing the Interval of Integration. If you suspect the validity of the approximation of an integral, subdivide the interval of integration into two or more subintervals, integrate the function over each subinterval, then add the resulting approximations. This causes the function to be evaluated at a new set of sample points, more likely revealing any previously hidden spikes. If the initial approximation is valid, it equals the sum of the approximations over the subintervals.

Example: Subdividing the Interval of Integration. Consider again the integral

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

Approximate the integral by subdividing the interval of integration into three subintervals, one from 0 to 10, the second from 10 to 100, and the third from 100 to 100,000.

First, integrate between 0 and 10. If you are still in the Integration application, simply supply the new value for *ULIM*.

10 ULIM
∫

```
∫=0.9995
LLIM ULIM ACC           ∫
```

The answer is very close to 1. Now integrate between 10 and 100.

10 LLIM
100 ULIM
∫

```
∫=0.0005
LLIM ULIM ACC           ∫
```

The answer is very close to 0. The sum of the approximations over the two subintervals is 1. Finally, integrate between 100 and 100,000. (Stay in the Integration application after executing this calculation. You will integrate this function again in the next section.)

100 LLIM
100000 ULIM
∫

```
∫=0.0000
LLIM ULIM ACC           ∫
```

The integral over the third subinterval is 0. The sum of the integrals over the three subintervals is 1.

Conditions That Prolong Calculation Time

In the first example in the preceding section, the algorithm gave an incorrect answer because it never detected the spike in the function $f(x) = xe^{-x}$. This happened because the variation in the function was too quick relative to the width of the interval of integration. In the second example, you obtained a very good approximation by subdividing the interval of integration into three subintervals between 0 and 100,000. However, for this function, there is a range of intervals that is small enough to obtain the correct answer, yet result in a very long calculation time.

Example: An Upper-Limit Approximation That Prolongs Calculation Time. Consider again the integral

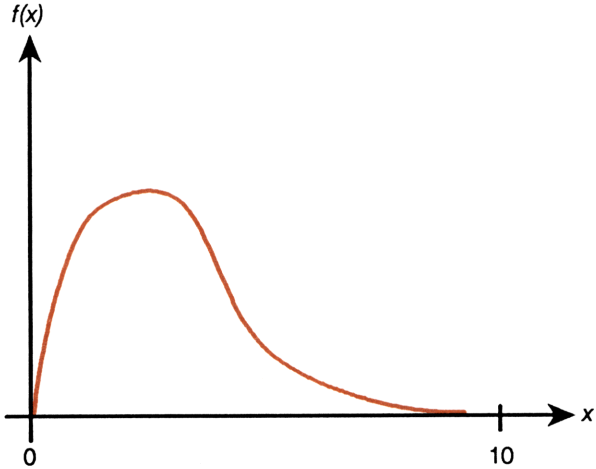
$$\int_0^{\infty} xe^{-x}\,dx$$

Approximate the integral by calculating it over the interval (0, 1,000).

Enter the new values for *LLIM* and *ULIM*. Then integrate with respect to *x*.

0 `LLIM` 1000 `ULIM` `∫`	`∫=1.0000` `LLIM ULIM ACC ∫`

This is the correct answer, but it took a long time to calculate. To understand why, compare the graph of the function between $x = 10$ and $x = 10^3$ (which looks about the same as that shown on page 141) with the following graph of the function between $x = 0$ and $x = 10$.



You can see that the function is "interesting" only at small values of x. At greater values of x, the function is not interesting since it decreases smoothly and gradually in a predictable manner.

The algorithm samples the function at increasing numbers of sample points until it has sufficient information about the function to provide an approximation that changes insignificantly when further samples are considered. In the previous section, when you evaluated the integral between 0 and 10, the algorithm needed to sample the function only at values where it was interesting but relatively smooth. The sample points, after the first few iterations, contributed no new information about the behavior of the function and the algorithm stopped.

In the last example, most of the sample points capture the function in the region where its slope is not varying much. The algorithm finds that the few sample points at small values of x return values of the function that change appreciably from one iteration to the next. Consequently, the function has to be evaluated at additional sample points before the disparity between successive approximations becomes sufficiently small.

For the integral to be approximated with the same accuracy over the larger interval as over the smaller interval, the density of sample points must be the same in the region where the function is interesting. To achieve the same density of sample points, the total number of sample points required over the larger interval is much greater than the number required over the smaller interval. Consequently, several more iterations are required over the larger interval to achieve an approximation of the same accuracy, and the calculation requires considerably more time.

5

Matrices

This chapter builds on material introduced to you in chapter 14 of your owner's manual. The following topics are covered:

- Using the matrix editor and indexing functions.
- Vector solutions.
- Solving simultaneous equations.
- Using the Solver with simultaneous equations.
- Matrix operations in programs.

Using the Matrix Editor and Indexing Functions

In the following example, you'll:

- Create a matrix.
- Use the matrix editor to manipulate data.
- Use indexing functions and statistics functions interactively.

Example: Accumulating Meteorological Data. Dr. Steven Stormwarning, noted meteorologist, has accumulated the following data and wishes to store it in a matrix in the HP-42S.

Day #	Temp	Wind	Humid
1	67	8	54
2	69	14	36
3	74	4	72

Creating a Named Matrix

Create a 4 × 4 matrix "*WTHR*".

4 [ENTER] ■[MATRIX] [▼] DIM
[ENTER] WTHR [ENTER]

```
x: 4.0000
DOT  CROSS  UVEC  DIM  INDEX  EDITN
```

Using the Matrix Editor

Enter the matrix editor and select the matrix you just created.

EDITN WTHR

```
1:1=0.0000
←  OLD  ↑  ↓  GOTO  →
```

Fill element 1:1 with the Alpha string DAY #. (Remember, to execute [ASTO], press [STO] in ALPHA mode.)

■[ALPHA] DAY # [ASTO] [.]
ST X [EXIT]

```
1:1="DAY #"
←  OLD  ↑  ↓  GOTO  →
```

Fill the remaining elements in row 1 with the corresponding Alpha strings from the table. (The keystrokes for element (1:2) are shown here.)

→ ■[ALPHA] TEMP [ASTO]
[.] ST X [EXIT] ...

```
1:4="HUMID"
←  OLD  ↑  ↓  GOTO  →
```

Now fill the remaining elements with the corresponding data.

→ 1 →
67 → 8
→ 54 →
2 → 69
→ 14 →
36 → 3
→ 74 →
4 → 72

```
4:4=72_
 ←  OLD  ↑  ↓  GOTO  →
```

Stormwarning finds that his assistant has incorrectly recorded the temperature on day 1; it was 77, not 67.

GOTO 2 [ENTER] 2 [ENTER] 77 [EXIT]

```
x: 77.0000
 DOT  CROSS  UVEC  DIM  INDEX  EDITN
```

Several days later the doctor has more data to add: on day #4, the temperature is 77, the windspeed is 5, and the humidity is 76. First, set the calculator to Grow mode to create a new row in the matrix.

EDITN WTHR ←
[▼] GROW [▲]
→

```
5:1=0.0000
 ←  OLD  ↑  ↓  GOTO  →
```

Fill in the new data.

4 → 77
→ 5 → 76

```
5:4=76_
 ←  OLD  ↑  ↓  GOTO  →
```

Stormwarning now realizes he has entered the data for day #5, not day #4. For day #4, the temperature was 68, the windspeed was 12, and the humidity was 41. First change the value in element 5:1 to 5.

← ←
← 5

```
5:1=5_
 ←  OLD  ↑  ↓  GOTO  →
```

Now insert the new row.

[▼] INSR

```
5:1=0.0000
 INSR    DELR    WRAP  GRO■
```

Enter the actual data for day #4.

[▲] 4 → 68
→ 12 → 41

```
5:4=41_
 ←  OLD  ↑  ↓  GOTO  →
```

Exit from the Matrix application.

[EXIT] [EXIT]

```
Y: 4.0000
X: 41.0000
```

Using Indexing Utilities and Statistics Functions Interactively

Dr. Stormwarning now wants to execute statistical operations on segments of his accumulated data. He would like to find the mean temperature and windspeed for the five days. He'll execute GETM to create in the X-register a 5×2 submatrix that contains the temperature and windspeed data. He'll then execute $\Sigma+$ to store the data from this submatrix in the summation (statistical) registers, select the STAT menu, and find the mean. (Remember that the $\Sigma+$ function *automatically* stores the data from an n-row $\times$ 2-column matrix into the currently defined summation registers. Refer to the discussion of the $\Sigma+$ function in chapter 15 of your owner's manual for more information.)

Specify *WTHR* as the indexed matrix.

■[MATRIX] [▼] INDEX WTHR

```
X: 41.0000
 DOT  CROSS UVEC  DIM  INDEX EDITN
```

Set the index pointers to element 2:2 (the first temperature data entry).

2 [ENTER] [▼] STOIJ

```
X: 2.0000
STOIJ RCLIJ STOEL RCLEL PUTM GETM
```

Now get the 5×2 submatrix that contains the temperature and windspeed data.

5 [ENTER] 2 GETM

```
X: [ 5x2 Matrix ]
STOIJ RCLIJ STOEL RCLEL PUTM GETM
```

Clear the summation registers, then store the data from the matrix in the summation registers. (If the calculator returns the message Nonexistent, the current SIZE allocation is insufficient.)

■ CLEAR CLΣ
■ TOP.FCN Σ+

```
x: 5.0000
STOIJ RCLIJ STOEL RCLEL PUTM GETM
```

Select the STAT menu and find the mean of the temperature data.

■ STAT MEAN

```
x: 73.0000
Σ+ SUM MEAN WMN SDEV CFIT
```

Find the mean of the windspeed data.

x≷y

```
x: 8.6000
Σ+ SUM MEAN WMN SDEV CFIT
```

The mean temperature for the five days is 73. The mean windspeed is 8.6.

Exit from the STAT menu.

EXIT

```
y: 73.0000
x: 8.6000
```

Matrix Utilities

The following routines use existing matrix functions to build useful matrix utilities.

Finding the Column Sum of a Matrix. CSUM calculates the column sum of the matrix in the X-register. (The column sum of a matrix *A* is a row matrix, each element of which is the sum of the elements of the corresponding column of matrix *A*.) The resultant matrix is returned to the X-register.

```
00 { 14-Byte Prgm }
01 LBL "CSUM"
02 TRANS
03 RSUM
```

```
04 TRANS
05 END
```

Finding the Column Norm of a Matrix. CNRM calculates the column norm of the matrix in the X-register. (The column norm of a matrix *A* is the maximum value (over all columns) of the sums of the absolute values of all elements in a column.) The result is returned to the X-register.

```
00 { 12-Byte Prgm }
01 LBL "CNRM"
02 TRANS
03 RNRM
04 END
```

Finding the Conjugate of a Complex Matrix. To find the conjugate of a complex matrix:

1. Place the matrix in the X-register.
2. Press ■ [COMPLEX].
3. Press [+/-].
4. Press ■ [COMPLEX].

The conjugate is returned to the X-register.

Finding the Matrix Sum of a Matrix. MSUM calculates the matrix sum (the sum of all the elements) of the matrix in the X-register. The result is returned to the X-register.

```
00 { 18-Byte Prgm }
01 LBL "MSUM"
02 XEQ "CSUM"
03 RSUM
04 DET
05 END
```

Finding the Maximum and Minimum Elements of a Matrix. MINMAX finds the maximum or minimum element of the *real* matrix in the X-register. The element is returned to the X-register. The indexed location of the element is returned to the Y- and Z-registers (column number in Y, row number in Z). Set flag 09 to find the maximum element. Clear flag 09 to find the minimum element.

Program:	Comments:
`00 { 61-Byte Prgm }` `01 LBL "MINMAX"`	

`02 STO "MINMAX"`
`03 INDEX "MINMAX"`
`04 RCLEL`
`05 GTO 03`
	Lines 02–05: Store the matrix currently in the X-register in *MINMAX*, index *MINMAX*, and establish element 1:1 as the current maximum or minimum element.

`06 LBL 01`
`07 RCLEL`
`08 FS? 09`
`09 GTO 02`
`10 X≥Y?`
`11 GTO 04`
`12 GTO 03`
	Lines 06–12: If flag 09 is clear, test if the current element is greater than the current minimum. If yes, go to label 04 (to maintain the current minimum). If no, go to label 03 (to make the current element the new minimum).

`13 LBL 02`
`14 X≤Y?`
`15 GTO 04`
	Lines 13–15: If flag 09 is set, test if the current element is less than the current maximum. If yes, go to label 04 (to maintain the current maximum). If no, make the current element the new maximum.

`16 LBL 03`
`17 RCLIJ`
`18 RCL ST Z`
`19 ENTER`
	Lines 16–19: Make the current element the new maximum or minimum.

`20 LBL 04`
`21 R↓`
`22 J+`
	Lines 20–24: Maintain the current maximum or minimum element.

```
23 FC? 77
24 GTO 01
25 END
```

Sorting a Matrix. SORT sorts the rows of the matrix in the X-register in ascending order by the values in column 1. The sorted matrix is returned to the X-register.

Program: **Comments:**

```
00 { 81-Byte Prgm }
01 LBL "SORT"

02 STO "SORTMAT"
03 INDEX "SORTMAT"

04 LBL 01
05 I+
06 FS? 76
07 GTO 04
08 RCLIJ
09 X<>Y
10 RCLEL
```

Lines 07 – 10: Establish the row number to sort. (On the first pass, row 2 is the row to sort, against row 1. On the second pass, row 3 is the row to sort, against rows 1 and 2.) Continue until all rows are sorted.

```
11 LBL 02
12 I-
13 RCLEL
14 FS? 76
15 GTO 03
16 X≤Y?
17 GTO 03
18 R↓
19 RCLIJ
20 RCL+ ST Y
21 R<>R
22 R↓
23 R↓
24 GTO 02
```

Lines 11 – 24: Successively move the "sort row" up the matrix until its column 1 value is greater than the column 1 value of the previous row.

```
25 LBL 03
26 R↓
27 R↓
28 1
29 STOIJ
30 GTO 01
31 LBL 04
32 RCL "SORTMAT"
```

Lines 25–32: Increment the "sort-row" number. If the increment causes the index pointer to wrap, return the sorted matrix to X and end the program.

```
33 END
```

Vector Solutions

Vectors are a special subset of matrices. You can describe a vector with either a 1-row × n-column matrix, or a 1-column × n-row matrix.

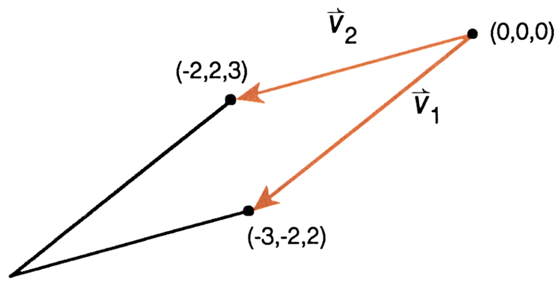
Geometry

The area of a parallelogram can be determined by the equation

$$A = \text{Frobenius norm (magnitude) of } (V\mathbf{V}_1 \times \mathbf{V}_2)$$

where $(\mathbf{V}_1 \times \mathbf{V}_2)$ is the vector cross product $\mathbf{V}_1$ and $\mathbf{V}_2$.

Example: The Area of a Parallelogram. Find the area of the following parallelogram.



Create vectors $\mathbf{V}_1$ and $\mathbf{V}_2$.

■ [MATRIX]
1 [ENTER] 3 [▼] DIM
[ENTER] V1 [ENTER]
1 [ENTER] 3 DIM
[ENTER] V2 [ENTER]

Enter values for each element in $\mathbf{V}_1$.

EDITN V1
3 [+/-] →
2 [+/-] →
2 [EXIT]

Enter values for each element in $\mathbf{V}_2$.

EDITN V2
2 [+/-] →
2 →
3 [EXIT]

Calculate the area.

[RCL] V1 [RCL] V2
CROSS ■ [CATALOG] FCN
[▼] ... FNRM

The area of the parallelogram is 15.0000.

Exit from the Matrix application.

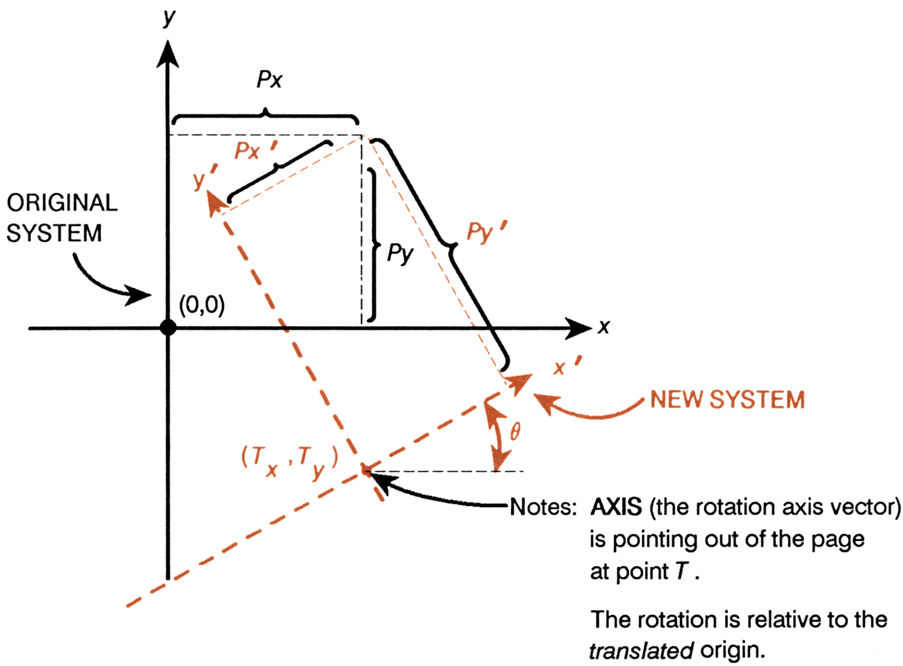
[EXIT]

```
Y: 3.0000
X: 15.0000
```

Coordinate Transformations

It is often necessary in dynamics or mechanical design problems to perform coordinate transformations. Coordinate transformations require you to:

- Calculate a unit vector.
- Add vectors.
- Calculate a vector dot product.
- Multiply vectors.
- Calculate a vector cross product.



The equation for a coordinate transformation of a point from the old system to a new system is

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

The equation for a coordinate transformation of a point from a new system to the old system is

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

where:

$\mathbf{P}'$ is the coordinates of the point in the new system.
$\mathbf{P}$ is the coordinates of the point in the old system.
$\mathbf{T}$ is the origin of the new system.
$\mathbf{n}$ is the unit vector of the axis about which the rotation is to be done.
θ is the rotation angle.

Note that the translation occurs before the rotation. The rotation is relative to the translated origin.

The following program, COORD, enables you to fill the vectors **P**, (or **P′**), **T**, and **AXIS** with data by programmatically invoking the matrix editor and enables you to specify either an old-to-new or new-to-old transformation. (**AXIS** is the rotation axis vector. COORD stores the data you supply for **AXIS** in the variable *n*, then calculates the unit vector **n**.)

To key in COORD: Create variables *P*, *T*, *P′*, *n*, and ∡ before program entry.

Here is an annotated listing of COORD.

Program:	**Comments:**
`00 { 216-Byte Prgm }` `01 LBL "COORD"`	

`02 EXITALL`
`03 CLMENU`
`04 "P"`
`05 KEY 1 GTO 01`
`06 "T"`
`07 KEY 2 XEQ 02`
`08 "AXIS"`
`09 KEY 3 XEQ 03`
`10 "∡"`
`11 KEY 4 XEQ 04`
	Lines 02 – 11: Build the main menu.

`12 LBL 98`
`13 MENU`
`14 STOP`
`15 GTO 98`
	Lines 12 – 15: Display the main menu.

`16 LBL 01`
`17 "P"`
`18 XEQ 99`
`19 "N→O"`
`20 KEY 5 GTO 05`
	Lines 16 – 22: Display the submenu to edit vector **P** (or **P′**) and choose the direction of the transformation.

```
21 "O→N"
22 KEY 6 GTO 06
```

```
23 LBL 97
24 MENU
25 CF 00
26 STOP
27 GTO 97
```

Lines 23–27: Display the submenu.

```
28 LBL 02
29 "T"
30 GTO 99
31 LBL 03
32 "n"
```

Lines 28–32: Place the vector names **T** and **n** in the Alpha register to create the vector.

```
33 LBL 99
34 CLMENU
35 ASTO ST L
36 1
37 ENTER
38 3
39 DIM IND ST L
40 EDITN IND ST L
41 "←"
42 KEY 1 XEQ 11
43 "→"
44 KEY 2 XEQ 12
45 KEY 9 GTO "COORD"
46 RTN
```

Lines 33–46: Create a 1 × 3 vector **P**, **T**, or **n** and open it for editing. Build matrix editor menu labels and prompt for data input.

```
47 LBL 11
48 ←
49 RTN
50 LBL 12
51 →
52 RTN
```

Lines 47–52: Execute the matrix editor functions.

```
53 LBL 04
54 INPUT "∡"
55 RTN
```

Lines 53–55: Prompt for the value of ∡.

```
56 LBL 05
57 SF 00
```

Lines 56 – 57: Set flag 00 for a new-to-old transformation.

```
58 LBL 06
59 EXITALL
60 RCL "P"
61 FC? 00
62 RCL- "T"
63 STO "P'"
64 RCL "n"
65 UVEC
66 STO "n"
67 DOT
68 1
69 RCL "∡"
70 COS
71 -
72 RCL× "n"
73 ×
74 RCL "∡"
75 COS
76 RCL× "P'"
77 +
78 RCL "P'"
79 RCL "n"
80 CROSS
81 RCL "∡"
82 FS? 00
83 +/-
84 SIN
85 ×
86 +
87 FS? 00
88 RCL+ "T"
89 STO "P"
90 GTO 01
```

Lines 58 – 90: Evaluate the transformation equation. If flag 00 is clear, calculate the old-to-new transformation. If flag 00 is set, calculate the new-to-old transformation.

```
91 END
```

To use COORD:

1. Press [XEQ] COORD.
2. Press T, then supply values for the elements of **T** using the matrix editor labels in the menu. Press [EXIT] to return to the main menu.
3. Press AXIS, then supply values for the elements of the rotation axis using the matrix editor labels in the menu. Press [EXIT] to return to the main menu. Note that COORD stores the rotation axis in variable *n*, calculates the unit vector of the rotation axis, and stores the unit vector back in *n*. If you press AXIS after executing a three-dimensional transformation, you will see the newly calculated elements of the unit vector, not the original rotation axis.

   For a two-dimensional transformation, set the rotation axis to (0, 0, 1).
4. Press ∡, then supply a value for ∡ and press [R/S].
5. Press P, then supply values for the elements of **P** (or **P′**) using the matrix editor labels in the menu. Then press O→N to convert from the old system to the new system, or press N→O to convert from the new system to the old system. The calculation is now executed.

Example: A Three-Dimensional Translation with Rotation. A three-dimensional coordinate system is translated from (0, 0, 0) to (2.45, 4.00, 4.25). After the translation, a 62.5° rotation occurs about the (0, −1, −1) axis. In the original system, a point had the coordinates (3.90, 2.10, 7.00). What are the coordinates of the point in the translated, rotated system?

For this problem:

P = (3.90, 2.10, 7.00)
T = (2.45, 4.00, 4.25)
AXIS = (0, −1, −1)
∡ = 62.5°

Set the display format to FIX 2. Set the angular mode to Degrees. Execute program COORD.

Keys	Display
■ [DISP] FIX 02 ■ [MODES] DEG [XEQ] COORD	x: 0.00 P T AXIS ∡

Enter the elements of **T**.

Keys	Display
T 2.45 → 4 → 4.25 → [EXIT]	x: 2.45 P T AXIS ∡

Enter the elements of the rotation axis.

Keys	Display
AXIS → 1 [+/-] → 1 [+/-] → [EXIT]	x: 0.00 P T AXIS ∡

Enter the value of ∡.

Keys	Display
∡ 62.5 [R/S]	x: 62.50 P T AXIS ∡

Enter the elements of **P**.

Keys	Display
P 3.9 → 2.1 → 7 →	1:1=3.90 ← → N→O O→N

Calculate the transformation.

Keys	Display
O→N	1:1=3.59 ← → N→O O→N

Element 1:1 of **P**′ is 3.59. Check element 1:2.

Keys	Display
→	1:2=0.26 ← → N→O O→N

Check element 1:3.

→

```
1:3=0.59
 ←  →  N→O O→N
```

The coordinates of the point in the new system are (3.59, 0.26, 0.59). Exit from program COORD and return the display format to FIX 4.

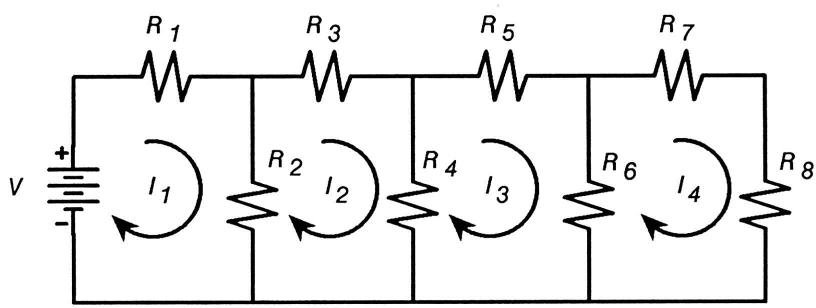
[EXIT] [EXIT]
■ [DISP] FIX 4 [ENTER]

```
Y: 1.0000
X: 0.5891
```

Solving Simultaneous Equations

Evaluation of an electrical circuit by the technique of loop currents generates a system of simultaneous equations. The number of equations in the system is equal to the number of loops in the circuit. The first example in this section finds the currents in a four-loop, purely resistive circuit (the terms in the system of equations are real numbers). The second example finds the currents in a four-loop circuit that has complex impedances (the terms in the system of equations are complex numbers).

Example: Solving Real-Number Simultaneous Equations. Consider the following four-loop circuit.



Apply the technique of loop currents to find the currents I_1, I_2, I_3, I_4.

The equations to be solved are (in variable form):

1. $(R_1 + R_2)(I_1) - (R_2)(I_2) = V$
2. $-(R_2)(I_1) + (R_2 + R_3 + R_4)(I_2) - (R_4)(I_3) = 0$
3. $-(R_4)(I_2) + (R_4 + R_5 + R_6)(I_3) - (R_6)(I_4) = 0$
4. $-(R_6)(I_3) + (R_6 + R_7 + R_8)(I_4) = 0$

Put the equations in matrix form, substituting the following values for the variables: V = 34 V and R_1 through R_8 = 1 Ω.

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 34 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Select the Simultaneous Equation application, and specify the number of unknowns.

■ [MATRIX] SIMQ 4 [ENTER]

```
x: 0.0000
MATA MATB MATX
```

Enter the values for the elements of the coefficient matrix *MATA*. (The keystrokes for the entering the first row data are shown here.) After entering all the values, return to the main menu.

MATA 2 →
1 [+/-] →
0 →
0 → ...

[EXIT]

```
x: 3.0000
MATA MATB MATX
```

Enter values for the constant matrix *MATB*.

MATB
34 ↓
0 ↓
0 ↓
0 [EXIT]

```
x: 0.0000
MATA MATB MATX
```

Calculate the unknowns.

MATX

```
1:1=21.0000
 ←  OLD  ↑  ↓  GOTO  →
```

I_1 is 21 A. Now check I_2.

↓

```
2:1=8.0000
 ←  OLD  ↑  ↓  GOTO  →
```

Check I_3.

↓

```
3:1=3.0000
 ←  OLD  ↑  ↓  GOTO  →
```

Check I_4.

↓

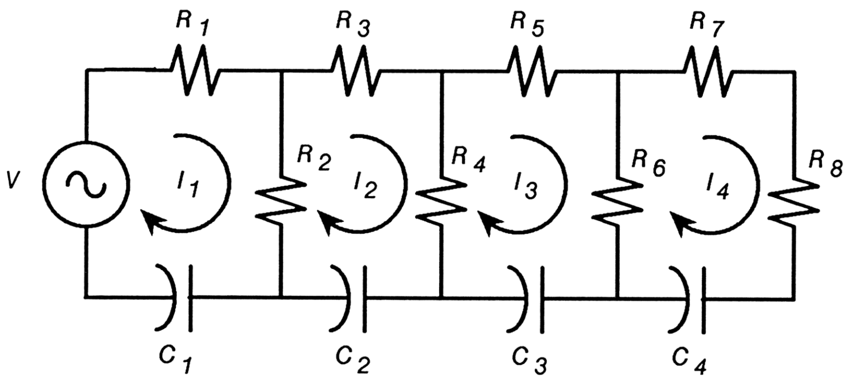
```
4:1=1.0000
 ←  OLD  ↑  ↓  GOTO  →
```

Leave the matrix editor. (Stay in the Simultaneous Equation application for the next example.)

[EXIT]

```
x: 1.0000
MATA MATB MATX
```

Example: Solving Simultaneous Equations That Have Complex Terms. Now consider the following circuit.



The capacitor in each loop of the circuit introduces a complex term into each loop equation:

1. $\left[(R_1 + R_2) - i(\frac{1}{\omega C_1})\right](I_1) - (R_2)(I_2) = V$

2. $-(R_2)(I_2) + \left[R_2 + R_3 + R_4 - i(\frac{1}{\omega C_2})\right](I_2) - (R_4)(I_3) = 0$

3. $-(R_4)(I_2) + \left[R_4 + R_5 + R_6 - i(\frac{1}{\omega C_3})\right](I_3) - (R_6)(I_4) = 0$

4. $-(R_6)(I_3) + \left[R_6 + R_7 + R_8 - i(\frac{1}{\omega C_4})\right](I_4) = 0$

Put the equations in matrix form, substituting the following values for the variables: V = 34 V, R_1 through R_8 = 5 Ω, ω = 100 radians/second, and C_1 through C_4 = 1 F.

$$\begin{bmatrix} 10 - i0.01 & -5 & 0 & 0 \\ -5 & 15 - i0.01 & -5 & 0 \\ 0 & -5 & 15 - i0.01 & -5 \\ 0 & 0 & -5 & 15 - i0.01 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 34 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Set the coordinate mode to Rectangular. Make *MATA* a complex matrix.

■ [MODES] RECT
0 [ENTER] ■ [COMPLEX]
[STO] [×] MATA

```
x: 0.0000 i0.0000
MATA MATB MATX
```

Enter the values for the elements of the matrix. (The keystrokes for the entering the first row data are shown here.) After entering all values, return to the main menu.

MATA
10 [ENTER] .01 [+/-]
■ [COMPLEX] →
5 [+/-] →
0 →
0 → ...

[EXIT]

```
x: 15.0000 -i0.0100
MATA MATB MATX
```

Solve for *MATX*. (*MATB* has the same value as in the previous example.)

MATX

```
1:1=4.2000 i0.0061
← OLD ↑ ↓ GOTO →
```

I_1 is 4.2000 + i0.0061 A. Now check I_2.

↓

```
2:1=1.6000 i0.0037
← OLD ↑ ↓ GOTO →
```

Check I_3.

↓

```
3:1=0.6000 i0.0019
 ←  OLD  ↑  ↓  GOTO  →
```

Check I_4.

↓

```
4:1=0.2000 i0.0008
 ←  OLD  ↑  ↓  GOTO  →
```

Exit from *MATX*.

[EXIT]

```
x: 0.2000 i0.0008
MATA MATB MATX
```

Make *MATA* and *MATX* real matrices. Exit from the Matrix application.

[RCL] MATA ■ [COMPLEX]
[STO] MATA
[RCL] MATX ■ [COMPLEX]
[STO] MATX
[EXIT] [EXIT]

```
y: [ 4x1 Matrix ]
x: [ 4x1 Matrix ]
```

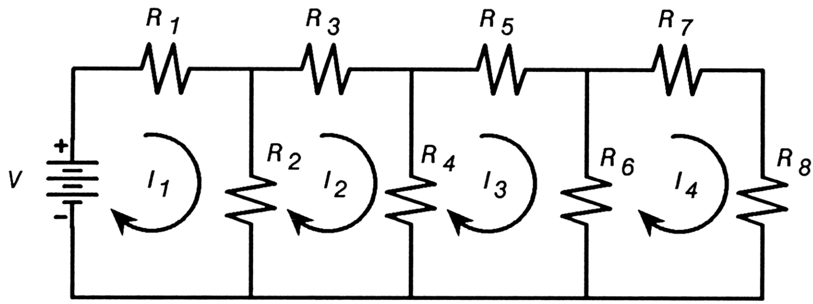
Using the Solver with Simultaneous Equations

In the examples in the previous section, you found the loop currents I_1 through I_4 by dividing the constant matrix *MATB* by the coefficient matrix *MATA*. You were limited in that example to solving specifically for the loop currents in the solution matrix *MATX*.

In the following example, you'll use the Solver and matrix division to find the value of one element of the *coefficient matrix*, *MATA*, given:

- Values for the other elements of the coefficient matrix.
- Values for the elements of the constant matrix.
- A specified relationship between two values of the solution matrix.

Example. Using the Solver to Find the Value of an Element of the Coefficient Matrix. Consider again the circuit from the previous section in this chapter.



Find the resistor value R_1 such that loop current I_1 is *20 A greater* than loop current I_2 ($I_1 = I_2+20$), when $V = 40$ V, and R_2 through $R_8 = 1\,\Omega$.

These conditions generate the following matrix equation.

$$\begin{bmatrix} R & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} I_2 + 20 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 40 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Part 1. Write the program for the Solver.

Program: / **Comments:**

```
00 { 82-Byte Prgm }
01 LBL "SIMUL"
```

```
02 MVAR "R"
03 MVAR "ROW"
04 MVAR "COL"
05 MVAR "D"
```

Lines 02–05: Declare the variables *R*, *ROW*, *COL*, and *D*.

```
06 INDEX "MATA"
07 RCL "ROW"
08 RCL "COL"
09 STOIJ
10 RCL "R"
11 STOEL
```

Lines 06–11: Index the coefficient matrix, and set the index pointer to the element specified by the current values of *ROW* and *COL* (lines 05–08). Store the current value of *R* (supplied first by you as initial guesses, and then iteratively by the Solver) in the specified element (lines 09–10).

```
12 RCL "MATB"
13 RCL÷ "MATA"
14 STO "MATX"
```

Line 12–14: Solve for *MATX*. *MATA* has the current value of *R* in the specified element.

```
15 INDEX "MATX"
16 RCLEL
17 I+
18 RCLEL
19 RCL+ "D"
20 -
```

Lines 15–20: Index the just-calculated solution matrix (line 14). Calculate $I_1 - (I_2 + D)$ (lines 15–20). The Solver iteratively supplies values for *R* until $I_1 - (I_2 + D) = 0$.

```
21 END
```

Part 2. Enter the Matrix application, and specify a system of equations with four unknowns.

■ [MATRIX] SIMQ 4 [ENTER]

```
x: 0.0000
MATA MATB MATX
```

Fill *MATA* with the known coefficients. Element 1:1 contains the unknown resistor value *R*. You can leave this element at its current value. (The keystrokes for the first two rows are shown here.) After entering all the data, return to the main menu.

MATA →
1 [+/-] →
→ →
1 [+/-] →
3 →
1 [+/-] → ...

[EXIT]

```
x: 3.0000
MATA MATB MATX
```

Fill *MATB* with the known constants, then exit from the Matrix application.

MATB 40 ↓
0 ↓
0 ↓
0 ↓
[EXIT] [EXIT] [EXIT]

```
Y: 0.0000
x: 40.0000
```

Select the Solver application, and then program SIMUL.

■ [SOLVER] SIMUL

```
x: 40.0000
R ROW COL D
```

Specify element 1:1 of the coefficient matrix.

1 ROW 1 COL

```
COL=1.0000
R ROW COL D
```

Enter 20 for *D*.

20 D

```
D=20.0000
R ROW COL D
```

Enter guesses of 0 and 10 for R and solve for R.

0 R
10 R
R

```
R=1.6190
 R  ROW  COL  D
```

Verify that element 1:1 of the coefficient matrix (R) is 1.6190.

MATRIX ▼
EDITN MATA

```
1:1=1.6190
 ←  OLD  ↑  ↓  GOTO  →
```

$R_1 = R - R_2 = 0.619\ \Omega$. Check the values for I_1 and I_2.

EXIT EDITN MATX

```
1:1=32.3077
 ←  OLD  ↑  ↓  GOTO  →
```

I_1 is 32.3077 A. Check I_2

↓

```
2:1=12.3077
 ←  OLD  ↑  ↓  GOTO  →
```

I_2 is 12.3077 A. Exit from the Matrix application.

EXIT EXIT

```
Y: 1.6190
X: 12.3077
```

Matrix Operations in Programs

All matrix functions except GOTO are programmable. The programs for advanced statistical operations in the following chapters use matrices extensively.

The program LIST on pages 176–178 enables you to accumulate statistical data in a matrix with the same keystroke sequence that you use in normal data entry into the summation registers.

The program MLR on pages 186–192 uses matrix and statistical functions to calculate a linear regression for data sets of three independent variables. MLR creates a coefficient matrix *MATA* and a constant matrix *MATB*. It executes matrix editor functions to fill them with data, then executes matrix division to calculate the solution matrix *MATX*.

The program PFIT on pages 218 – 222 plots the statistical data from the matrix currently in the X-register, then fits and plots a curve to the data using the current statistical model. It plots the curve and the data points using x-y data pairs from complex matrices.

6

Statistics

This chapter presents five programs for statistical operations. The programs use statistical functions introduced in chapter 15 of your owner's manual, and integrate matrix operations presented in the previous chapter and in chapter 14 of your owner's manual.

- Three programs enable you to accumulate data in a matrix for subsequent statistical operations:
  - LIST enables you to fill an $n \times 2$ matrix $\Sigma LIST$ with x- y data pairs with the same keystroke sequence that you use to enter data into the summation registers.
  - ΣFORM stores an $n \times m$ matrix in $\Sigma LIST$ and redimensions $\Sigma LIST$ to $nm \times 2$. Each element of the original matrix becomes an element of column 2 of $\Sigma LIST$. Column 1 is filled with zeros.
  - XVALS fills column 1 of $\Sigma LIST$ with x-values 1, 2, 3, ... , n for linear or exponential curve fitting.
- MLR calculates a multiple linear regression for two or three independent variables using the Σ+ function and matrix operations.
- PFIT plots the x-y data pairs from $\Sigma LIST$ and uses FCSTY to plot a curve to the data according to the currently selected statistical model. (The annotated listing of PFIT is in chapter 7 on pages 218–222.)

List Statistics

To supply a set of *x-y* data pairs to the calculator for subsequent statistical operations, you use the keystroke sequence

y-value [ENTER] *x-value* [Σ+]

for each data pair. The summation coefficients in the 6 (or 13) summation registers are automatically recalculated each time you press [Σ+]. The calculator does *not*, however, maintain a list of the individual data pairs.

To update the summation registers *and* maintain a list of the *x-y* data pairs, you:

1. Create a 2-column matrix.
2. Use matrix editor functions to fill the matrix with the data pairs.
3. Place the matrix in the X-register.
4. Execute Σ+ to accumulate the data in the summation registers.

(You did this in chapter 5 in the section "Using Indexing Utilities and Statistics Functions Interactively".)

The LIST Program. The following program, LIST, enables you to fill a 1- or 2-column matrix *ΣLIST* with *x-y* data pairs using the keystroke sequence

y-value [ENTER] *x-value* LIST+ (for each data pair).

where LIST+ is one of three menu keys built by LIST. Note that this is the same keystroke sequence that you use to enter statistical data into the summation registers.

To key in LIST:

1. Create variable *ΣLIST* before program entry.
2. Assign functions J+ and J− to the CUSTOM menu before program entry.
3. Create labels LIST, LIST+, LIST−, and CLIST when you begin program entry.

Here is an annotated listing of LIST.

Program:	**Comments:**
`00 { 197-Byte Prgm }` `01 LBL "LIST"`	

`02 CLMENU`
`03 "LIST+"`
`04 KEY 1 XEQ "LIST+"`
`05 "LIST-"`
`06 KEY 2 XEQ "LIST-"`
`07 "CLIST"`
`08 KEY 6 XEQ "CLIST"`
`09 MENU`
`10 STOP`
`11 GTO "LIST"`
	Lines 02–11: Build and display the menu keys.

```
12 LBL "LIST+"
13 SF 25
14 XEQ I
15 FC?C 25
16 GTO 02
17 GROW
18 J-
19 J+
20 WRAP
```

Lines 12 – 20: If Σ*LIST* exists, index it and make it grow by one row. If it doesn't exist, create and index it (in lines 32 – 42).

```
21 LBL 00
22 STOEL
23 FS? 01
24 GTO 01
25 J+
26 X<>Y
27 STOEL
28 X<>Y
```

Lines 21 – 28: Store the *x*-value into the matrix. If flag 01 is clear, then also store the *y*-value.

```
29 LBL 01
30 VIEW "ΣLIST"
31 RTN
```

Lines 29 – 31: View the Σ*LIST* matrix.

```
32 LBL 02
33 1
34 FS? 01
35 1
36 FC? 01
37 2
38 DIM "ΣLIST"
39 XEQ I
40 R↓
41 R↓
42 GTO 00
```

Lines 32 – 42: Create the 1- or 2-column matrix Σ*LIST*.

```
43 LBL "LIST-"
44 SF 25
45 XEQ I
46 FC? 25
47 RTN
48 J-
49 RCLEL
50 FS? 01
51 GTO 03
52 J-
53 RCLEL
```

Lines 43 – 53: Recall the element(s) in the last row of Σ*LIST* to the X- (or X- and Y-) register(s).

```
54 LBL 03
55 DELR
56 FS?C 25
57 GTO 01
```

Lines 54 – 57: Delete the last row of Σ*LIST*.

```
58 LBL "CLIST"
59 CLV "ΣLIST"
60 RTN
```

Subroutine CLIST, lines 58 – 60: Clear the variable Σ*LIST*.

```
61 LBL I
62 INDEX "ΣLIST"
63 RTN
```

Subroutine I, lines 61 – 63: Index Σ*LIST*.

```
64 END
```

To use LIST:

1. For two-variable statistics (x- and y-values), clear flag 01. For one-variable statistics (x-values only), set flag 01; the program makes $\Sigma LIST$ a 1-column matrix.
2. Press [XEQ] LIST .
3. Clear $\Sigma LIST$ by pressing CLIST.
4. Enter data pairs by pressing *y-value* [ENTER] *x-value* LIST+ (for each data pair).
5. You can delete the last data pair by pressing LIST-.

Example: Accumulating Statistical Data in a Matrix. Use program LIST to accumulate the following x-y data pairs in the matrix $\Sigma LIST$. Then find the mean of the x- and y-values.

x-value	y-value
6	2
5	3
9	5
12	6
21	11
7	4

Clear flag 01 for two-variable statistics. Start LIST.

■ [FLAGS] CF 01
[XEQ] LIST

```
x: 0.0000
LIST+ LIST-  CLIST
```

Clear $\Sigma LIST$.

CLIST

```
x: 0.0000
LIST+ LIST-  CLIST
```

Enter the first data pair.

2 [ENTER] 6 LIST+

```
ΣLIST=[ 1x2 Matrix ]
LIST+ LIST-  CLIST
```

Key in the next data pair.

3 [ENTER] 5 LIST+

```
ΣLIST=[ 2x2 Matrix ]
LIST+ LIST-  CLIST
```

Key in the remaining data pairs (the keystrokes are not shown here). Exit from LIST.

[EXIT]

```
Y: 4.0000
X: 7.0000
```

Clear the summation registers. Recall *ΣLIST* to the X-register.

■ [CLEAR] CLΣ
[RCL] ΣLIST

```
Y: 7.0000
X: [ 6x2 Matrix ]
```

Accumulate the data from *ΣLIST* into the summation registers.

[Σ+]

```
Y: 7.0000
X: 6.0000
```

Find the mean of the x- and y-values.

■ [STAT] MEAN

```
X: 10.0000
Σ+ SUM MEAN WMN SDEV CFIT
```

The mean of the x-values is 10. Check the mean of the y-values.

[x⥂y]

```
X: 5.1667
Σ+ SUM MEAN WMN SDEV CFIT
```

Exit from the STAT menu.

[EXIT]

```
Y: 10.0000
X: 5.1667
```

Redimensioning the *ΣLIST* Matrix to *nm* × 2. In the previous example, you used LIST to create a 6 × 2 matrix *ΣLIST*. You then recalled *ΣLIST* to the X-register, and executed Σ+ to accumulate the *x-y* data pairs from the matrix into the summation registers. *To execute* Σ+ *when a matrix is in the X-register, that matrix must have a column dimension equal to 2.* If, for example, you use LIST to create an $n \times 1$ matrix *ΣLIST* (by setting flag 01), you must redimension it before executing Σ+.

The following program, ΣFORM, redimensions *any* matrix *ΣLIST* of dimension $n \times m$ to dimension $nm \times 2$. All of the elements in the input matrix are moved to the second column. The first column is filled with 0's (zeros).

```
00 { 58-Byte Prgm }
01 LBL "ΣFORM"

02 2
03 RCL "ΣLIST"
04 DIM?
05 ×
06 DIM "ΣLIST"
07 INDEX "ΣLIST"
08 1
09 ENTER
10 2
11 R<>R
12 RCL "ΣLIST"
13 TRANS
14 STO "ΣLIST"

15 END
```

Filling Column Two of *ΣLIST* with Evenly Spaced Integers. You may want to fit a linear or exponential curve to a set of one-variable statistical data. The following program, XVALS, fills the first column of the *ΣLIST* matrix with integers 1, 2, 3, ..., *n*. If *ΣLIST* is a 1-column matrix, XVALS automatically creates the new column.

Program: **Comments:**

```
00 { 46-Byte Prgm }
01 LBL "XVALS"
```

```
02 RCL "ΣLIST"
03 DIM?
04 1
05 -
06 X≤0?
07 XEQ "ΣFORM"
08 INDEX "ΣLIST"
```

Lines 02–08: Recall Σ*LIST*. If it is a 1-column matrix, execute ΣFORM to make it a 2-column matrix. Then index it.

```
09 LBL 00
10 RCLIJ
11 X<>Y
12 ↓
13 FC? 76
14 GTO 00
```

Lines 9–14: Fill column 1 with integers 1, 2, 3, ..., n. Continue to the end of the column.

```
15 END
```

Using the Summation-Coefficient Functions (Σ+, Σ-, and CLΣ) in Programs

The program MLR in this section uses the Σ+ function and matrix operations to calculate a multiple linear regression for three independent variables.

For a set of data points $\{ (x_i, y_i, z_i, t_i), i = 1, 2, ..., n \}$, MLR fits a linear equation of the form

$$t = a + bx + cy + dz$$

by the least squares method.

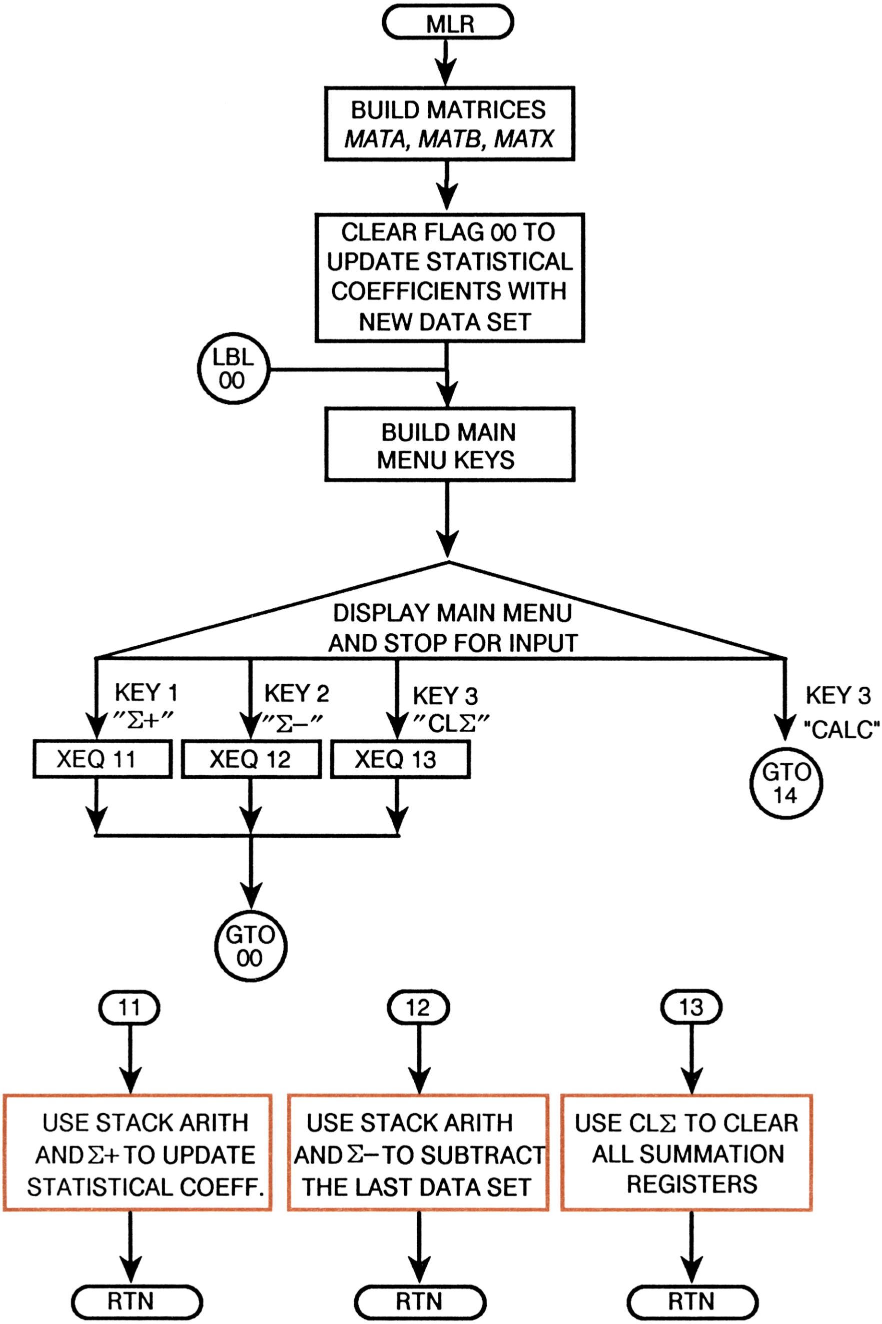
Regression coefficients a, b, c, and d are calculated by solving the following set of equations.

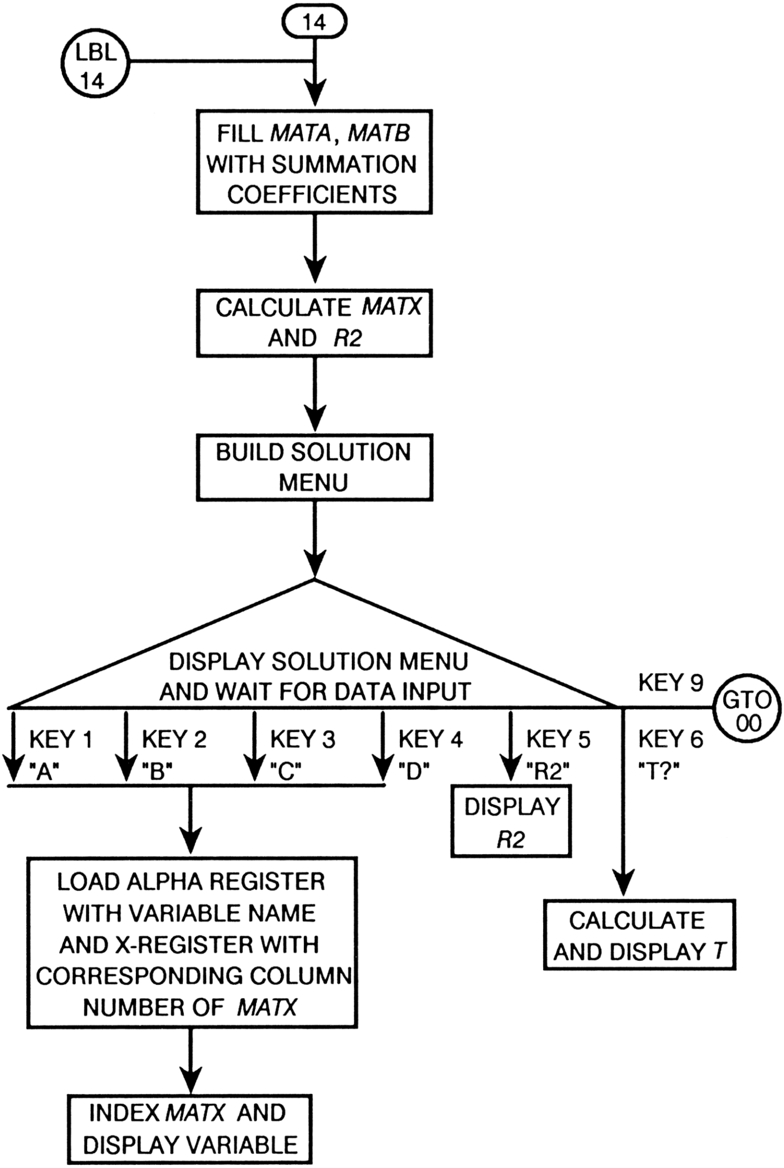
$$\begin{bmatrix} n & \Sigma x_i & \Sigma y_i & \Sigma z_i \\ \Sigma x_i & \Sigma (x_i)^2 & \Sigma x_i\, y_i & \Sigma x_i\, z_i \\ \Sigma y_i & \Sigma y_i\, x_i & \Sigma (y_i)^2 & \Sigma y_i\, z_i \\ \Sigma z_i & \Sigma z_i\, x_i & \Sigma z_i\, y_i & \Sigma (z_i)^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \Sigma t_i \\ \Sigma x_i\, t_i \\ \Sigma y_i\, t_i \\ \Sigma z_i\, t_i \end{bmatrix}$$

The coefficient of determination R^2 is defined as

$$R^2 = \frac{a\Sigma t_i + b\Sigma x_i\, t_i + c\Sigma y_i\, t_i + d\Sigma z_i\, t_i - \frac{1}{n}(\Sigma t_i)^2}{\Sigma (t_i^2) - \frac{1}{n}(\Sigma t_i)^2}$$

Here is a flowchart for MLR.


MLR
BUILD MATRICES
MATA, MATB, MATX
CLEAR FLAG 00 TO UPDATE STATISTICAL COEFFICIENTS WITH NEW DATA SET
LBL 00
BUILD MAIN MENU KEYS
DISPLAY MAIN MENU AND STOP FOR INPUT
KEY 1 "Σ+"
KEY 2 "Σ–"
KEY 3 "CLΣ"
KEY 3 "CALC"
XEQ 11
XEQ 12
XEQ 13
GTO 14
GTO 00
11
12
13
USE STACK ARITH AND Σ+ TO UPDATE STATISTICAL COEFF.
USE STACK ARITH AND Σ– TO SUBTRACT THE LAST DATA SET
USE CLΣ TO CLEAR ALL SUMMATION REGISTERS
RTN
RTN
RTN


14
LBL 14
FILL MATA, MATB WITH SUMMATION COEFFICIENTS
CALCULATE MATX AND R2
BUILD SOLUTION MENU
DISPLAY SOLUTION MENU AND WAIT FOR DATA INPUT
KEY 9
GTO 00
KEY 1 "A"
KEY 2 "B"
KEY 3 "C"
KEY 4 "D"
KEY 5 "R2"
KEY 6 "T?"
DISPLAY R2
LOAD ALPHA REGISTER WITH VARIABLE NAME AND X-REGISTER WITH CORRESPONDING COLUMN NUMBER OF MATX
CALCULATE AND DISPLAY T
INDEX MATX AND DISPLAY VARIABLE

To key in MLR:

1. Assign functions →, ↑, ←, ↓, I− and J+ to the CUSTOM menu before program entry.
2. Create variables *MATA*, *MATB*, *MATX*, *R2*, and *T* before program entry.

Here is an annotated listing of the program.

Program:	**Comments:**
`00 { 460-Byte Prgm }` `01 LBL "MLR"`	

`02 REALRES`
`03 4`
`04 ENTER`
`05 1`
`06 DIM "MATX"`
`07 DIM "MATB"`
`08 4`
`09 ENTER`
`10 DIM "MATA"`
`11 CF 00`
`12 LINΣ`
	Lines 02 – 12: Set to calculate real results only. Create 4 × 1 matrices *MATX* and *MATB*. Create 4 × 4 matrix *MATA*. Clear flag 00 (set to Σ+ mode). Set to Linear (statistics) mode (calculate six summation coefficients).

`13 LBL 00`
`14 CF 21`
`15 CLMENU`
`16 "Σ+"`
`17 KEY 1 XEQ 11`
`18 "Σ-"`
`19 KEY 2 XEQ 12`
`20 "CLΣ"`
`21 KEY 3 XEQ 13`
`22 "CALC"`
`23 KEY 6 GTO 14`
`24 MENU`
`25 CLD`
`26 STOP`
`27 GTO 00`
	Lines 13 – 27: Build and display the menu keys Σ+, Σ-, CLΣ, and CALC.

```
28 LBL 11
29 RCL× ST Z
30 FS? 00
31 +/-
32 STO+ 13
33 CLX
34 LASTX
35 RCL× ST T
36 FS? 00
37 +/-
38 STO+ 15
39 CLX
40 LASTX
41 R↓
42 RCL× ST Y
43 FS? 00
44 +/-
45 STO+ 14
46 CLX
47 LASTX
48 RCL× ST Z
49 FS? 00
50 +/-
51 STO+ 16
52 CLX
53 LASTX
54 R↓
55 ΣREG 07
56 FS? 00
57 RTN
58 Σ+
```

Subroutine 11, lines 28–58: *Emulate* Σ+ (or Σ− if flag 00 set) to update the following summation coefficients: Σxz in R_{13}, Σxt in R_{15}, Σyz in R_{14}, Σyt in R_{16}. Execute Σ+ to update the following coefficients: Σz in R_{07}, Σz^2 in R_{08}, Σt in R_{09}, Σt^2 in R_{10}, Σzt in R_{11}, n in R_{12}.

```
59 LBL 01
60 CLX
61 LASTX
62 R↓
63 R↓
64 ΣREG 01
65 FS?C 00
66 RTN
67 Σ+
68 RTN
```

Subroutine 01, lines 59 – 68: Execute Σ+ to update the following coefficients: Σx in R_{01}, Σx^2 in R_{02}, Σy in R_{03}, Σy^2 in R_{04}, Σxy in R_{05}, and n in R_{06}. (Note that n is also calculated in subroutine 11.)

```
69 LBL 12
70 SF 00
71 XEQ 11
72 Σ-
73 XEQ 01
74 Σ-
75 RTN
```

Subroutine 12, lines 69 – 75: Emulate Σ– (set flag 00) to update the coefficients calculated in subroutine 11. Execute Σ– to update the remaining coefficients.

```
76 LBL 13
77 ΣREG 11
78 CLΣ
79 ΣREG 07
80 CLΣ
81 ΣREG 01
82 CLΣ
83 RTN
```

Subroutine 13, lines 76 – 83: Execute CLΣ to clear all defined summation registers.

```
84 LBL 14
85 "Calculating"
86 AVIEW
87 0
88 STO× "MATA"
89 INDEX "MATA"
90 J+
91 RCL 01
92 ÷
93 RCL 03
```

Lines 84 – 147, calculation of coefficients a, b, c, d, and R^2: Fill *MATA* with x, y, z summation coefficients. Fill *MATB* with t summation coefficients. Calculate *MATX* (*MATB* ÷ *MATA*). Calculate R^2.

```
94 →
95 RCL 07
96 ↓
97 RCL 13
98 ←
99 RCL 05
100 ↓
101 J+
102 RCL 14
103 ↓
104 RCL "MATA"
105 TRANS
106 STO+ "MATA"
107 RCL 08
108 ↑
109 ←
110 RCL 04
111 ↑
112 ←
113 RCL 02
114 ↑
115 ←
116 RCL 06
117 STOEL
118 INDEX "MATB"
119 RCL 09
120 ↓
121 RCL 15
122 ↓
123 RCL 16
124 ↓
125 RCL 11
126 STOEL
127 RCL "MATB"
128 RCL÷ "MATA"
129 STO "MATX"
130 LASTX
131 TRANS
132 X<>Y
133 ×
134 FNRM
```

```
135 RCL 09
136 X↑2
137 RCL÷ 06
138 -
139 LASTX
140 RCL 10
141 X<>Y
142 -
143 ÷
144 STO "R2"
145 CLD
146 FS? 55
147 SF 21
```

```
148 LBL 02
149 "A"
150 KEY 1 XEQ 21
151 "B"
152 KEY 2 XEQ 22
153 "C"
154 KEY 3 XEQ 23
155 "D"
156 KEY 4 XEQ 24
157 "R2"
158 KEY 5 XEQ 25
159 "T?"
160 KEY 6 XEQ 26
161 KEY 9 GTO 00
162 MENU
163 STOP
164 GTO 02
```

Lines 148 – 164: Build and display the solution menu.

```
165 LBL 21
166 1
167 "a"
168 GTO 03
169 LBL 22
170 2
171 "b"
```

Subroutines 21 – 25, lines 165 – 192: Display the calculated coefficients *a*, *b*, *c*, *d*, and *R2*. If PRON has been executed, print the coefficients (lines 187 and 191).

```
172 GTO 03
173 LBL 23
174 3
175 "c"
176 GTO 03
177 LBL 24
178 4
179 "d"
180 LBL 03
181 1
182 INDEX "MATX"
183 STOIJ
184 RCLEL
185 ⊢"="
186 ARCL ST X
187 AVIEW
188 RTN
189 LBL 25
190 RCL "R2"
191 VIEW "R2"
192 RTN
```

Subroutine 26, lines 193 – 205: Forecast *T* based on the calculated coefficients *a*, *b*, *c*, and *d*. Display *T* and, if PRON has been executed, print *T*.

```
193 LBL 26
194 INDEX "MATX"
195 XEQ 04
196 XEQ 04
197 XEQ 04
198 +
199 +
200 I-
201 RCLEL
202 +
203 STO "T"
204 VIEW "T"
205 RTN
```

```
206 LBL 04
207 Σ-
208 RCLEL
209 RCL× ST T
210 RTN

211 END
```

Subroutine 04, lines 206–210: Calculate terms *bx*, *cy*, and *dz*.

To use MLR:

1. Press [XEQ] MLR
2. Press CLΣ to clear the summation registers.
3. Enter each data set, using the keystroke sequence *t-value* [ENTER] *z-value* [ENTER] *y-value* [ENTER] *x-value* Σ+ .
4. Press CALC .
5. Press the corresponding menu keys to see the values of variables *a*, *b*, *c*, *d*, and *R2*.
6. To forecast *T*, use the keystroke sequence *z-value* [ENTER] *y-value* [ENTER] *x-value* T? .
7. To return to the main menu, press [EXIT].

Example: A Linear Regression For Three Independent Variables. Find the regression equation for the following set of data.

i	1	2	3	4	5
x_i	7	1	11	11	7
y_i	25	29	56	31	52
z_i	6	15	8	8	6
t_i	60	52	20	47	33

Execute MLR.

[XEQ] MLR

```
x: 4.0000
Σ+  Σ-  CLΣ        CALC
```

Clear the summation registers. Enter the first data set, starting with the t-value.

CLΣ
60 [ENTER] 6 [ENTER] 25 [ENTER]
7 Σ+

```
x: 1.0000
Σ+ Σ- CLΣ       CALC
```

Enter the second data set.

52 [ENTER] 15 [ENTER] 29 [ENTER]
1 Σ+

```
x: 2.0000
Σ+ Σ- CLΣ       CALC
```

Enter the remaining data sets (the keystrokes are not shown here). Now calculate the regression coefficients and the coefficient of determination.

CALC

```
x: 0.9989
A B C D R2 T?
```

Check the value of a.

A

```
a=103.4473
A B C D R2 T?
```

Check the value of b.

B

```
b=-1.2841
A B C D R2 T?
```

Check the value of c.

C

```
c=-1.0369
A B C D R2 T?
```

Check the value of d.

D

```
d=-1.3395
A B C D R2 T?
```

Check the value of R^2.

R2

```
R2=0.9989
A B C D R2 T?
```

Calculate T (the forecasted value of t given values for x, y, and z). Use the values from data set #4.

8 [ENTER] 31 [ENTER] 11
T?

```
T=46.4616
 A  B  C  D  R2  T?
```

(The actual value of t in data set #4 is 47.) Return to the main menu and clear the statistics registers for new data.

[EXIT] CLΣ

```
x: 46.4616
 Σ+  Σ-  CLΣ       CALC
```

Exit from MLR.

[EXIT]

```
y: 11.0000
x: 46.4616
```

Curve Fitting in Programs

The curve fitting functions FCSTX, FCSTY, SLOPE, YINT, CORR, LINF, LOGF, EXPF, PWRF, and BEST are programmable.

Refer to program PFIT on pages 218–222 in the following chapter. PFIT uses FCSTY in line 89 to forecast a y-value based on the currently selected statistical model for each of 110 x-values. A curve is then plotted with the 110 data pairs.

7

Graphics and Plotting

The following topics are covered in this chapter:

- Building graphics patterns.
- Multifunction plotting.
- Plotting statistical data from a complex matrix.

Graphics

The program HPLOGO in this section uses the XTOA and AGRAPH functions to build the Hewlett-Packard company logo in the center of the display.

To key in HPLOGO:

1. Assign the functions XTOA, CLA, ARCL, and XEQ to the CUSTOM menu.
2. Create the variable *BLOCK*.

Here is the annotated listing.

Program:	**Comments:**
00 { 441-Byte Prgm } 01 LBL "HPLOGO"	
02 CLLCD 03 CF 34 04 CF 35	Lines 02 – 04: Clear the display for graphics. Clear flags 34 and 35 so that graphics placed in the display with AGRAPH are merged with any graphics already in the display. (The top and bottom halves of the logo are built separately and merged in the display.)
05 XEQ "TOP" 06 1 07 ENTER 08 40 09 AGRAPH	Lines 05 – 09: Call subroutine TOP to build the top half of the logo. Then display the top half of the logo, starting at pixel (1, 40).
10 XEQ "BOT" 11 9 12 ENTER 13 40 14 AGRAPH 15 RTN	Lines 10 – 15: Call subroutine BOT to build the bottom half of the logo. Then display the bottom half of the logo, starting at pixel (9, 40).
16 LBL "TOP" 17 CLA 18 255 19 XTOA 20 XTOA 21 XTOA 22 XTOA 23 XTOA 24 XTOA 25 ASTO "BLOCK" 26 CLA 27 254	Subroutine TOP, lines 16 – 91: Build the Alpha string that represents the top half of the logo. (Begin by building the Alpha string that represents an 8 × 6 block of on-pixels and storing that string in the variable *BLOCK*.)

```
28 XTOA
29 ARCL "BLOCK"
30 255
31 XTOA
32 63
33 XTOA
34 15
35 XTOA
36 7
37 XTOA
38 XTOA
39 3
40 XTOA
41 1
42 XTOA
43 129
44 XTOA
45 224
46 XTOA
47 120
48 XTOA
49 62
50 XTOA
51 39
52 XTOA
53 161
54 XTOA
55 224
56 XTOA
57 96
58 XTOA
59 0
60 XTOA
61 1
62 XTOA
63 129
64 XTOA
65 225
66 XTOA
```

```
67 97
68 XTOA
69 33
70 XTOA
71 XTOA
72 35
73 XTOA
74 163
75 XTOA
76 231
77 XTOA
78 103
79 XTOA
80 15
81 XTOA
82 31
83 XTOA
84 63
85 XTOA
86 ARCL "BLOCK"
87 255
88 XTOA
89 254
90 XTOA
91 RTN
```

Subroutine BOT, lines 92–156: Build the Alpha string that represents the bottom half of the logo.

```
92 LBL "BOT"
93 CLA
94 127
95 XTOA
96 ARCL "BLOCK"
97 255
98 XTOA
99 252
100 XTOA
101 240
102 XTOA
103 224
104 XTOA
```

```
105 XTOA
106 192
107 XTOA
108 198
109 XTOA
110 135
111 XTOA
112 129
113 XTOA
114 0
115 XTOA
116 XTOA
117 6
118 XTOA
119 7
120 XTOA
121 129
122 XTOA
123 224
124 XTOA
125 120
126 XTOA
127 30
128 XTOA
129 7
130 XTOA
131 5
132 XTOA
133 132
134 XTOA
135 XTOA
136 XTOA
137 198
138 XTOA
139 199
140 XTOA
141 225
142 XTOA
143 224
```

```
144 XTOA
145 240
146 XTOA
147 248
148 XTOA
149 252
150 XTOA
151 ARCL "BLOCK"
152 255
153 XTOA
154 127
155 XTOA
156 RTN

157 END
```

Example: Building a Logo. Display the Hewlett-Packard logo. If you have a printer, modify HPLOGO to print the logo. Then print it.

Execute HPLOGO.

[XEQ] HPLO

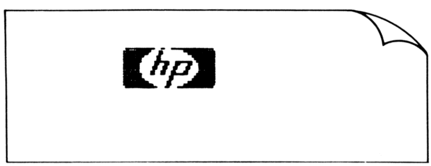
Insert the instruction PRLCD after line 14 of HPLOGO to print the logo.

■[PRGM] ■[GTO] [·] 14 [ENTER]
[▼] PRLCD [EXIT]

```
Y: 9.0000
X: 40.0000
```

Print the logo.

■[PRINT] [▲] PON
[XEQ] HPLO



Using Binary Data to Build a Graphics Pattern. To build the logo in the previous example, you had to calculate the column print number for each of 91 columns – a time-consuming effort. The following program, BINDATA, calculates the column print number when you input the equivalent sequence of binary numbers in a column pattern.

Program:	**Comments:**
`00 { 60-Byte Prgm }` `01 LBL "BINDATA"`	

`02 CF 34`
`03 CF 35`
`04 BINM`
	Lines 02 – 04: Clear flags 34 and 35. Set the calculator to Binary mode.

`05 LBL 00`
`06 CLX`
`07 STOP`
`08 "     "`
`09 126`
`10 X<>Y`
`11 X>Y?`
`12 GTO 01`
`13 """`
`14 XTOA`
`15 ├""  "`
	Lines 05 – 15: Clear the X-register and suspend program execution for binary data entry (lines 06 – 07). Build an Alpha string of five spaces (line 08). Test if the binary data (converted to decimal form) is greater than 126. If so, go to label 01. If not, enclose the corresponding HP-42S Alpha character in quotes and append two spaces to the Alpha register.

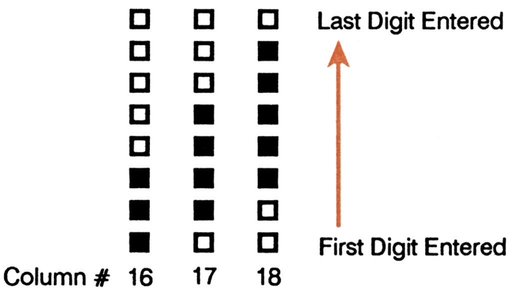
`16 LBL 01`
`17 AIP`
`18 AVIEW`
`19 CLA`
`20 XTOA`
`21 1`
`22 ENTER`
`23 66`
`24 AGRAPH`
`25 GTO 00`
	Lines 16 – 25: Append the number in the X-register (the decimal equivalent of the binary data) to the Alpha register and display the current contents of the Alpha register (lines 17 – 18). (The Alpha register contains the decimal number equivalent of the binary data. If that number is less than 128, the Alpha register also contains the corresponding HP-42S character, enclosed in quotes). Build the equivalent column pattern and display it, beginning at pixel (1, 66) (lines 20 – 24). Return to label 00 for the next data entry (line 25).

`26 END`	

To use BINDATA:

1. An on-pixel has value 1. An off-pixel has value 0.
2. Enter digits beginning at the bottom of the column.
3. If, for example, you enter only six digits, the bottom two digits are interpreted to be zeros.
4. Press [R/S] after data entry to see the calculation. After the calculation is displayed, simply key in the next sequence of numbers when you are ready.

Example: Using Binary Data to Build a Logo. Columns 16–18 of the Hewlett-Packard logo in the previous example have the following pixel patterns.



Use BINDATA to calculate the column print number for each column.

Start the program.

[XEQ] BINDA

x: 0
A...F HEX DEC OCT BIN LOGIC

Enter the binary data for column 16.

11100000 [R/S]

224

The column print number for column 16 is 224. There is no equivalent Alpha character. The column pattern is at the right of the display. Now enter the binary data for column 17.

01111000 [R/S]

```
"x"   120   ı
```

The column print number for column 17 is 120. The equivalent HP-42S character is "x". (You can therefore either accumulate 120 in the X-register and execute XTOA, *or* accumulate character "x" in the Alpha register. The column pattern is at the right of the display. Enter the binary data for column 18.

00111110 [R/S]

```
">"   62    ı
```

The column print number for column 18 is 62. The equivalent HP-42S Alpha character is ">". Now exit from the program.

[EXIT]

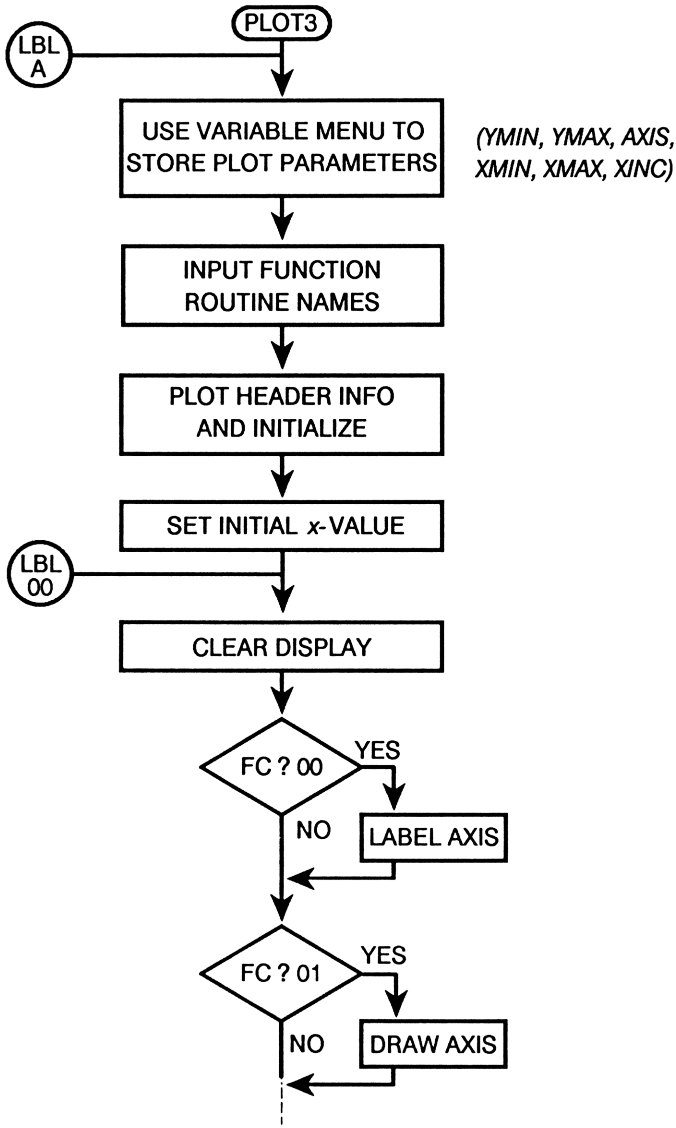
```
Y: 1.0000
X: 0.0000
```

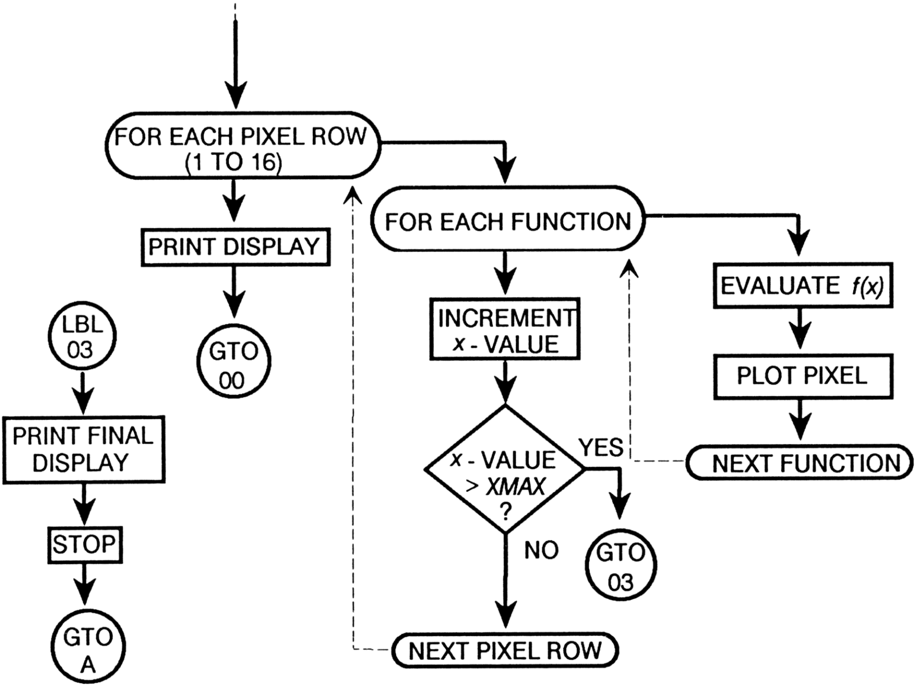
(Refer to the character table in your owner's manual (appendix E) and note that five of the first 127 characters cannot be typed from the HP-42S keyboard. The character codes are 4, 6, 13, 27, and 30. Program BINDATA shows you the character corresponding to each of these codes, but because these characters cannot be typed, you must accumulate the corresponding character code in the X-register and execute XTOA.)

Multifunction Plots

The program PLOT3 in this section enables you to plot up to three functions concurrently on the HP 82240A Infrared Printer. It is based on the program PLOT in the section "Example Programs" in chapter 10 of your owner's manual. As in PLOT, you supply to the program the name of the routine that defines the function you wish to plot. However, in PLOT3, you can supply up to three routine names.

Here is a flowchart for PLOT3.





To key in PLOT3: Create variables *YMIN*, *YMAX*, *AXIS*, *XMIN*, *XMAX*, *XINC*, *FCN1*, *FCN2*, and *FCN3* before program entry.

Here is an annotated listing of the program.

Program:	**Comments:**
`00 { 424-Byte Prgm }` `01 LBL "PLOT3"`	

`02 MVAR "YMIN"`
`03 MVAR "YMAX"`
`04 MVAR "AXIS"`
`05 MVAR "XMIN"`
`06 MVAR "XMAX"`
`07 MVAR "XINC"`
	Lines 02 – 07: Declare the menu variables.

```
08 LBL A
09 VARMENU "PLOT3"
10 CF 34
11 CF 35
12 CLA
13 STOP
14 EXITALL
```

Lines 08 – 14: Display the menu and suspend program execution for data input.

```
15 "FCN1"
16 XEQ 07
17 "FCN2"
18 XEQ 07
19 "FCN3"
20 XEQ 07
```

Lines 15 – 20: Prompt for the function names (the subroutine labels).

```
21 ADV
22 "Plot of:"
23 PRA
24 ADV
25 SF 12
26 RCL "FCN1"
27 XEQ 08
28 RCL "FCN2"
29 XEQ 08
30 RCL "FCN3"
31 XEQ 08
32 ADV
33 CF 12
34 PRV "YMIN"
35 PRV "YMAX"
36 PRV "AXIS"
37 PRV "XMIN"
38 PRV "XMAX"
39 PRV "XINC"
40 ADV
41 "← YMIN"
42 ⊢"       YMAX →␊"
43 PRA
```

Lines 21 – 43: Print the header information. (In line 42, there are seven spaces in the Alpha string before *YMAX*.)

```
44 130
45 RCL "YMAX"
46 RCL- "YMIN"
47 ÷
48 STO 00
```

Lines 44 – 48: Calculate the relative *y*-value of one pixel.

```
49 RCL "XMIN"
50 STO 01
```

Lines 49 – 50: Store the first *x*-value.

```
51 LBL 00
52 CLLCD
53 FC? 00
54 XEQ 05
55 FC? 01
56 XEQ 06
57 1.016
58 STO 02
```

Lines 51 – 58: Clear the display. If flag 00 is clear, label the *x*-axis. If flag 01 is clear, draw an axis. Build a loop counter corresponding to the 16 rows in the display.

```
59 LBL 01
60 49.051
61 STO 03
```

Lines 59 – 61: Build a loop counter for the three possible functions. (The character codes for characters "1", "2", and "3" are 49, 50, and 51 respectively. Routine 02 uses these numbers to create the variables.)

```
62 LBL 02
63 "FCN"
64 RCL 03
65 XTOA
66 ASTO ST X
67 RCL IND ST X
68 STR?
69 XEQ 04
70 ISG 03
71 GTO 02
72 RCL "XINC"
73 16
74 ÷
75 STO+ 01
```

Lines 62 – 83: Create the Alpha strings FCN1, FCN2, and FCN3 successively. Call each string to the X-register, then recall to the X-register the *variable* that matches that string. Test if the variable has an Alpha string (a function program name) in it (lines 62 – 68). If so, plot a pixel for each function. Increment the *x*-value. If the plot is complete (if *x*-value > *XMAX*), go to label 03. If the current display is complete (if rows 1 – 16 are filled), then print the display and start a new one.

```
76 RCL "XMAX"
77 RCL 01
78 X>Y?
79 GTO 03
80 ISG 02
81 GTO 01
82 PRLCD
83 GTO 00
```

```
84 LBL 03
85 PRLCD
86 RTN
87 GTO A
```

Lines 84 – 87: Print the final display and stop. (Line 87 enables you to restart the program by pressing [R/S].)

```
88 LBL 04
89 RCL 01
90 XEQ IND ST Y
91 SF 24
92 RCL- "YMIN"
93 RCL× 00
94 1
95 +
96 CF 24
97 RCL 02
98 X<>Y
99 X>0?
100 PIXEL
101 RTN
```

Subroutine 04, lines 88 – 101: Evaluate the function at x and plot the appropriate pixel.

```
102 LBL 05
103 CF 21
104 CLA
105 ARCL 01
106 AVIEW
107 SF 21
108 RTN
```

Subroutine 05, lines 102 – 108: Label the x-axis.

```
109 LBL 06
110 1
111 RCL "AXIS"
112 RCL- "YMIN"
113 RCL× 00
114 +/-
115 1
116 -
117 PIXEL
118 +/-
119 2
120 -
121 "xxxxx"
122 AGRAPH
123 RTN
```

Subroutine 06, lines 109 – 123: Draw the axis. (In line 120, the Alpha string is five "multiply" characters: press ■ ALPHA × × × × × ENTER.)

```
124 LBL 07
125 CF 21
126 ASTO ST L
127 CLX
128 AVIEW
129 PSE
130 CLA
131 FS? 55
132 SF 21
133 SF 25
134 RCL IND ST L
135 CF 25
136 STR?
137 ARCL ST X
138 AON
139 CLD
140 STOP
141 AOFF
142 ALENG
143 X≠0?
144 ASTO ST X
```

Lines 124 – 146: Prompt for an Alpha string (function name). If the variable already contains an Alpha string, that string is recalled to the Alpha register as the default.

```
145 STO IND ST L
146 RTN

147 LBL 08
148 CLA
149 STR?
150 ARCL ST X
151 STR?
152 PRA
153 RTN

154 END
```

Subroutine 08, lines 147–153: Print the function names.

To use PLOT3:

1. Execute PRON and turn on your Infrared Printer.
2. Write a routine for each function that you want to plot. The current *x*-value is in the X-register when the program calls the function routines. The routines need not recall the current *x*-value to the X-register.
3. Set the display format to ALL.
4. Start the program (press [XEQ] PLOT3).
5. Supply the plot parameters. For example, specify 20 for *YMIN* by pressing 20 YMIN.
6. After supplying values for the plot parameters, press [R/S].
   a. As prompted, store the name of each function routine in a function variable. For example, to supply the name TAN for *FCN1*, press TAN [R/S] at the first prompt.
   b. If you have already supplied a routine name for a function variable, that name is displayed at the prompt. If you want to leave that name in the variable, simply press [R/S].
   c. If you want to plot only two functions, supply names for only two variables. Leave the Alpha register clear for the third variable (just press [R/S] when prompted). If a name is displayed for the third variable, press [←] to clear the Alpha register, then press [R/S]. If you want to plot only one function, supply a name for only one variable and leave the Alpha register clear (or clear it) for the other two variables.

Example: Plotting Multiple Functions. Use PLOT3 to plot the following functions.

1. $y = \sin x$
2. $y = 0.35(\ln x)(\cos x)$

First, write routines to describe the functions.

```
00 { 9-Byte Prgm }
01 LBL "SINE"
02 SIN
03 END
```

Program:

```
00 { 27-Byte Prgm }
01 LBL "LNCOS"
02 COS
03 LASTX
04 0.0001
05 +
06 LN
07 0.35
08 ×
09 ×
10 END
```

Comments:

Lines 04–05: Ensure that the program does not attempt to execute ln (0).

Set the Display format to ALL. Execute PRON. Clear flags 00 and 01 to draw and label the *x*-axis. Start PLOT3.

■ DISP ALL
■ PRINT ▲ PON
■ FLAGS ■ FLAGS
CF 00 CF 01 EXIT
XEQ PLOT3

```
x: 0
YMIN YMAX AXIS XMIN XMAX XINC
```

Plot *y*-values between −3 and 3, and set the axis at $y = 0$.

3 +/- YMIN
3 YMAX
0 AXIS

```
AXIS=0
YMIN YMAX AXIS XMIN XMAX XINC
```

Plot x-values between 0 and 720 in increments of 60 per display.

0 XMIN
720 XMAX
60 XINC

```
XINC=60
YMIN YMAX AXIS XMIN XMAX XINC
```

Supply the program name SINE for the first function variable.

[R/S] SINE

```
SINE_
ABCDE FGHI JKLM NOPQ RSTUV WXYZ
```

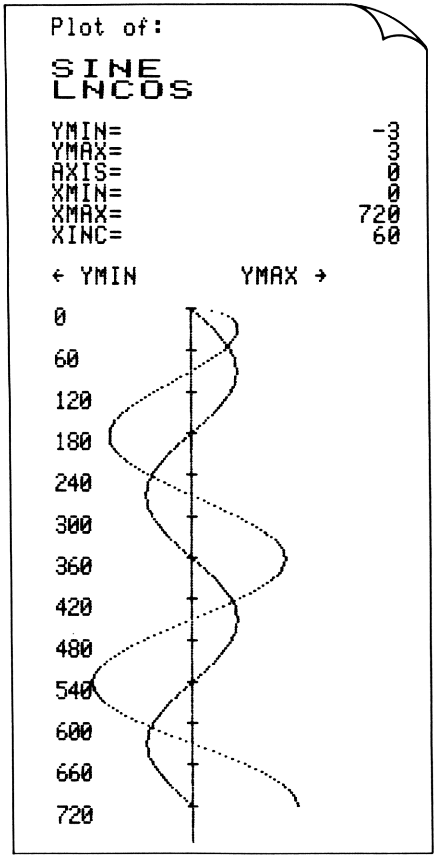
Supply the program name LNCOS for the second function variable.

[R/S] LNCOS

```
LNCOS_
ABCDE FGHI JKLM NOPQ RSTUV WXYZ
```

Leave the Alpha register clear for the third function variable and start the plot. The printer output is shown here.

[R/S] [R/S]



Exit from the program. Return the display format to FIX 4.

[EXIT]
[DISP] FIX 4 [ENTER]

Y: 720.0000
X: 723.7500

Plotting Data from a Complex Matrix

In previous programs, you have used:

- PIXEL to turn on individual pixels in the display. You specify the pixel number in the X- and Y-registers (row number in Y and column number in X).
- AGRAPH to display a graphics pattern. You specify the location of the pattern in the display by placing a pixel number in the X- and Y-registers (row number in Y and column number in X).

PIXEL and AGRAPH *operate* on the numbers in the X- and Y-registers.

The efficiency of these functions is enhanced by enabling them to operate on a *complex matrix* in the X-register, where each element of the complex matrix has the form

$$x\text{-value} + iy\text{-value}$$

When such a matrix is in the X-register, PIXEL turns on *each* pixel in the display as specified by the elements in the matrix. For example, consider the following complex matrix.

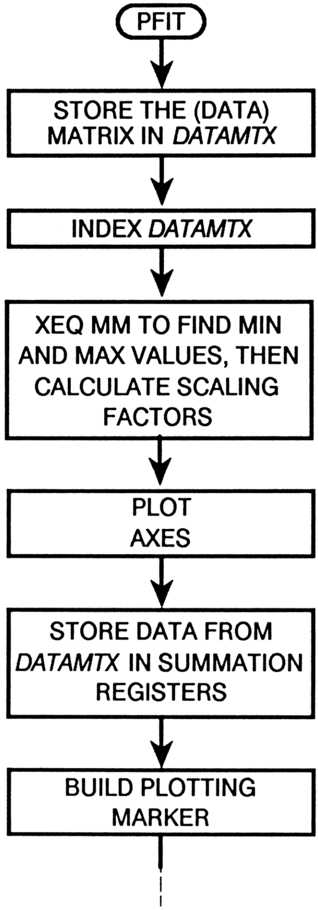
$$\begin{bmatrix} 1+i10 & 5+i20 \\ 10+i30 & 16+i40 \end{bmatrix}$$

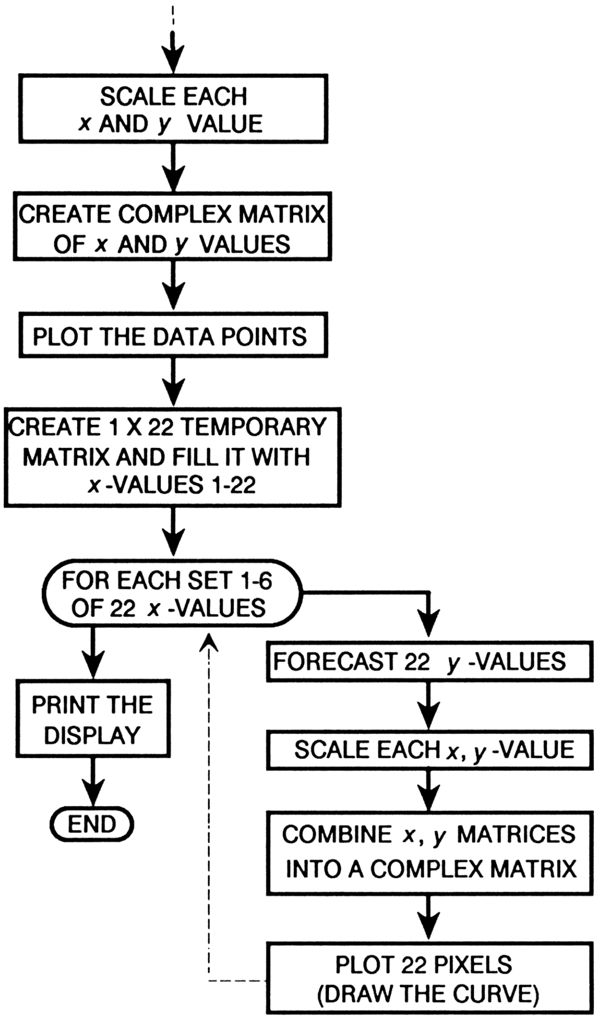
If you execute PIXEL when this matrix is in the X-register, pixels (1, 10), (5, 20), (10, 30), and (16, 40) are each turned on.

Similarly, AGRAPH places the graphics pattern that is encoded in the Alpha register at *each* position in the display as specified by the elements in the matrix.

Note that PIXEL and AGRAPH operate on the *rectangular* form of the complex matrix. Before entering numbers into the complex matrix, set the angular mode to Rectangular.

The program PFIT in this section plots the individual data pairs from the real $n \times 2$ matrix in the X-register, then fits and plots a curve to that data using the currently selected statistical model. PFIT creates one complex matrix and executes AGRAPH to mark each data point with a "+" character. PFIT then creates a second complex matrix and executes PIXEL to plot the forecasted curve.


PFIT
STORE THE (DATA) MATRIX IN *DATAMTX*
INDEX *DATAMTX*
XEQ MM TO FIND MIN AND MAX VALUES, THEN CALCULATE SCALING FACTORS
PLOT AXES
STORE DATA FROM *DATAMTX* IN SUMMATION REGISTERS
BUILD PLOTTING MARKER



To key in PFIT:

1. Create variable *DATAMTX* before program entry.
2. Create label MM when you begin program entry.

Here is an annotated listing of PFIT.

Program:	**Comments:**
`00 { 295-Byte Prgm }` `01 LBL "PFIT"`	

`02 CF 34`
`03 CF 35`
`04 RECT`
`05 STO "DATAMTX"`
`06 INDEX "DATAMTX"`
	Lines 02–06: Clear flags 34 and 35. Store the matrix that is in the X- register in *DATAMTX* and index *DATAMTX*.

`07 XEQ "MM"`
`08 STO 02`
`09 -`
`10 128`
`11 ÷`
`12 STO 01`
`13 STO÷ 02`
	Lines 07–13: Call subroutine MM to find the minimum and maximum *x*-values. Then calculate the *x*-value scaling factor.

`14 XEQ "MM"`
`15 X<>Y`
`16 STO 04`
`17 -`
`18 13`
`19 X<>Y`
`20 ÷`
`21 STO 03`
`22 STO× 04`
`23 2`
`24 STO- 04`
`25 STO- 02`
	Lines 14–25: Call subroutine MM to find the minimum and maximum *y*-values. Then calculate the *y*-value scaling factor.

```
26 CLLCD
27 RCL 04
28 X>0?
29 RCL- ST X
30 RCL 02
31 X>0?
32 CLX
33 PIXEL
```

Lines 26 – 33: Plot the axes.

```
34 ΣREG 11
35 CLΣ
36 RCL "DATAMTX"
37 Σ+
```

Lines 34 – 37: Store the data in *DATAMTX* into the summation registers.

```
38 CLA
39 2
40 XTOA
41 7
42 XTOA
43 X<>Y
44 XTOA
```

Lines 38 – 44: Build the "+" character (used to mark each data point in the plot).

```
45 RCL "DATAMTX"
46 TRANS
47 STO "DATAMTX"
48 INDEX "DATAMTX"
49 DIM?
50 1
51 X<>Y
52 GETM
53 DELR
54 RCL÷ 01
55 RCL- 02
```

Lines 45 – 55: Make the matrix $2 \times n$ and index it (lines 45 – 48). Make two $1 \times n$ matrices, where the matrix in the X- register is the x-values, and the matrix in *DATAMTX* is the y- values (lines 49 – 53). Convert the x-values to screen coordinates for plotting (lines 54 – 56).

```
56 RCL "DATAMTX"
57 RCL× 03
58 RCL- 04
59 COMPLEX
```

Lines 56 – 59: Convert the y-values to screen coordinates (lines 56 – 58). Convert the matrices in X and Y to one complex matrix in X, each element of which is: x-value + iy-value (line 57).

```
60 1
61 ENTER
62 COMPLEX
63 -
64 AGRAPH
```

Lines 60 – 64: Subtract 1 + i1 from each value (to set the center of the "+" character at the data point) (lines 60 – 63). Place the center of the "+" at the coordinates defined by each element of the matrix (plot the data points) (line 64).

```
65 RCL "REGS"
66 1
67 ENTER
68 22
69 DIM "REGS"
```

Lines 65 – 69: Recall the registers matrix to the X-register, and redimension it to a 1×22 temporary matrix.

```
70 21
71 LBL 01
72 STO IND ST X
73 DSE ST X
74 GTO 01
75 STO 00
```

Lines 70 – 75: Fill the temporary matrix with values 0 through 21.

```
76 R↑
77 RCL "REGS"
78 X<>Y
79 STO "REGS"
80 CLX
```

Lines 76 – 80: Store the data from the registers back into *REGS*, then clear the X-register.

```
81 6
82 X<>Y
83 1
84 +
```

Lines 81 – 84: Establish a loop counter 6.00000 in the Y-register. Change the values in the temporary matrix to 1 through 22. (These values represent the first set of 22 x-values.)

```
85 LBL 02
86 ENTER
87 RCL+ 02
88 RCL× 01
89 FCSTY
90 RCL× 03
91 RCL- 04
92 COMPLEX
93 PIXEL
94 COMPLEX
95 R↓
96 22
97 +
98 DSE ST Y
99 GTO 02
100 PRLCD
101 CLV "DATAMTX"
102 RTN
```

Lines 85 – 102: Forecast y-values for a set of 22 x-values, make a complex matrix of x-y data pairs, and plot each data pair. Repeat for five more sets of x-values.

```
103 LBL "MM"
104 RCLEL
105 ENTER
106 LBL 09
107 I+
108 FS? 76
109 RTN
110 RCLEL
111 X<Y?
112 X<>Y
113 X<>Y
114 R↓
115 X>Y?
```

Subroutine MM, lines 103 – 120: Find the maximum and minimum elements of one column of the matrix for scaling. At the start of the subroutine, the matrix *DATAMTX* is indexed with the index pointer at the top of a column. At the end of the subroutine, the minimum element of the column is in X, the maximum element is in Y, and the index pointer is at the top of the next column.

```
116 X<>Y
117 R↓
118 RCL ST Z
119 GTO 09
120 RTN

121 END
```

To use PFIT:

1. Select a statistical model. For example, press ■ [STAT] CFIT MODL LINF.
2. Place a 2-column real matrix of data pairs in the X-register.
3. Press [XEQ] PFIT

Example: Plotting Data from a Compression Process and Fitting a Power Curve to the Data. Many compression processes can be correlated using the power curve

$$P = aV^{-b}$$

where:

P is the pressure.
V is the volume.
$-b$ is the polytropic constant.

Enter the following pressure-volume data in the *ΣLIST* matrix. Then use PFIT to plot the data and to plot a power curve to the data.

V	*P*
10	210
30	40
50	12
70	9
90	6.8

Execute program LIST. (If you've deleted the program, you need to key it in again. The listing is in section "List Statistics" in chapter 6.)

[XEQ] LIST

```
x: 0.0000
LIST+ LIST- CLIST
```

Clear the $\Sigma LIST$ matrix, then fill it with the data.

CLIST
210 [ENTER] 10 LIST+
40 [ENTER] 30 LIST+
12 [ENTER] 50 LIST+
9 [ENTER] 70 LIST+
6.8 [ENTER] 90 LIST+

```
ΣLIST=[ 5x2 Matrix ]
LIST+ LIST- CLIST
```

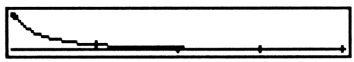
Exit from LIST. Recall $\Sigma LIST$ to the X-register.

[EXIT]
[RCL] ΣLIST

```
y: 90.0000
x: [ 5x2 Matrix ]
```

Set the statistical model to a power fit. Execute PRON if you have a printer. Execute PFIT.

■ [STAT] CFIT MODL PWRF
(■ [PRINT] [▲] PON)
■ [TOP.FCN] XEQ PFIT



Exit from PFIT. Check the correlation coefficient for the data.

[EXIT] CORR

```
x: -0.9939
FCSTX FCSTY SLOPE YINT CORR MODL
```

The correlation coefficient is −0.9939. Check the value of $-b$.

SLOPE

```
x: -1.6152
FCSTX FCSTY SLOPE YINT CORR MODL
```

The value of $-b$ is −1.6252. Exit from the STAT menu.

[EXIT] [EXIT]

```
y: -0.9939
x: -1.6152
```

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

N

O

P

Q

T

U

V

X

Programming Examples and Techniques for Your HP-42S Calculator

Programming Examples and Techniques contains examples in mathematics, science, engineering, and finance to help you more fully utilize the built-in applications in your HP-42S calculator. *Programmed* solutions are emphasized. Graphics and plotting with the HP 82240A Infrared Printer are also addressed.

- **Programming**
  Simple Programming • Branching • Controlled Looping • Indirect Addressing in Programs • Flags in Programs • Error Trapping
- **Enhancing HP-41 Programs**
  Using Named Variables • Using HP-42S Data Input and Output Functions • Operations with HP-42S Data Types • Using the Two-Line Display • Using Menu Variables • Assigning a Program to the CUSTOM Menu
- **The Solver**
  Basic Use of the Solver • Providing Initial Guesses for the Solver • Emulating the Solver • Using the Solver in Programs • More on How the Solver Works
- **Integration**
  Basic Integration • Approximating an Integral That Has an Infinite Limit • Using the Solver and Integration Interactively • More on How Integration Works
- **Matrices**
  Using the Matrix Editor and Indexing Functions • Vector Solutions • Solving Simultaneous Equations • Using the Solver with Simultaneous Equations • Matrix Operations in Programs
- **Statistics**
  List Statistics • Using the Summation-Coefficient Functions in Programs • Curve Fitting in Programs
- **Graphics and Plotting**
  Graphics • Multifunction Plots • Plotting Data from a Complex Matrix

HEWLETT PACKARD

Reorder Number
00042-90020

00042-90019 English
Printed in U.S.A. 7/88

0 88698 00036 6