HP 48
INSIGHTS
PART II: Problem-Solving Resources

William C. Wickes
HP 48 Insights II Program Disk

As a convenience for readers of HP 48 Insights II who use or have access to an IBM-compatible or Macintosh personal computer, Larken Publications is offering a disk containing all of the HP 48 programs and examples described in the book. By downloading the programs individually or collectively from your computer to your HP 48, you avoid the effort and errors of entering the programs manually from the calculator keyboard.

To order one or more of these disks, remove this page from your book, fill out the ordering information below, and send it with your payment to:

Larken Publications
Department PC
4517 NW Queens Ave.
Corvallis OR 97330 USA

Make checks payable to Larken Publications (no charge or C.O.D. orders). Foreign orders must be paid in U.S. Funds through a U.S. bank or via international postal money order.

Name

Address

City State Zip

Country

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Unit Price</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 48 Insights II Program Disk</td>
<td>$10.00</td>
<td></td>
</tr>
</tbody>
</table>

(Optional) Airmail postage outside of USA, Canada, Mexico 1.00

TOTAL $ 

Disk Type (check one): □ IBM 5.25" □ IBM 3.5" □ Macintosh 3.5"
HP 48 Insights

II. Problem-Solving Resources

William C. Wickes

Larken Publications
4517 NW Queens Avenue
Corvallis, Oregon 97330
Acknowledgements

I thank my wife, Susan, and my children, Kenneth and Lara, for their help in the preparation of this manuscript.
Dedicated to those mathematics teachers with the vision and courage to adopt calculators like the HP 48 for their classrooms, to improve the understanding and enthusiasm of their students.
## CONTENTS

13. Introduction to Part II. ................................................. 401
   13.1 About This Book .................................................. 402
   13.2 Notation .......................................................... 403
   13.3 Terminology ....................................................... 408

14. HP Solve ............................................................... 409
   14.1 The Equation Entry Menu ....................................... 412
   14.2 Basic HP Solve Operation ...................................... 414
      14.2.1 Independent, Dependent, and Unknown Variables 415
      14.2.2 An Example ................................................. 417
      14.2.3 Equation Management .................................... 420
         14.2.3.1 Using Subdirectories 420
         14.2.3.2 The Equation Catalog 421
   14.3 Solving With Units ............................................. 423
      14.3.1 Faster Solving with Units 425
      14.3.2 Monitoring Convergence ................................ 426
   14.4 Interpreting Results ........................................... 427
      14.4.1 Qualifying Messages .................................... 428
   14.5 First Guesses .................................................... 429
      14.5.1 How Many Guesses? ...................................... 430
      14.5.2 Examples Using \( x(x - 2)(x + 2) = 0 \) 431
   14.6 Obtaining Guesses .............................................. 433
   14.7 Finding Critical Points ....................................... 436
   14.8 Using ISOL with HP Solve ..................................... 439
   14.9 Programmable Solving ......................................... 440
   14.10 Secondary Results ............................................ 441
   14.11 Modifying the Solver Menu .................................. 443

15. Plotting ............................................................... 447
   15.1 The Plot Menus .................................................. 447
   15.2 Plotting Essentials ............................................ 450
      15.2.1 DRAW 452
         15.2.1.1 Autoscaling 453
      15.2.2 EQ and ΣDAT 453
      15.2.3 PPAR 454
         15.2.3.1 Saving A Graph 455
      15.2.4 The Plot Scale 456
         15.2.4.1 Redimensioning the Graph Screen 458
      15.2.5 The Independent Variable 459
      15.2.6 The Dependent Variable 460
17. Expression Manipulations ........................................... 541
  17.1 Extensive Manipulations ........................................ 541
    17.1.1 COLCT 545
    17.1.2 EXPAN 547
  17.2 The EquationWriter Subexpression Mode ...................... 550
    17.2.1 Navigating the Expression 552
    17.2.2 Editing Subexpressions 554
    17.2.3 RULES 556
      17.2.3.1 Repeated Operations 561
      17.2.3.2 Condensed RULES Notation 562
      17.2.3.3 Moving Terms 562
      17.2.3.4 Commutation 564
      17.2.3.5 Association 565
      17.2.3.6 Distribution 566
      17.2.3.7 Merging 567
      17.2.3.8 Prefix Operations 567
      17.2.3.9 Unit Identities 568
      17.2.3.10 Adding Fractions 569
      17.2.3.11 Logarithms 570
      17.2.3.12 Exponentials 570
      17.2.3.13 Definition Expansions 570
      17.2.3.14 Addition Angle Formulae 570
      17.2.3.15 Collecting Terms 572
  17.3 Pattern Matching and Substitution .......................... 572
  17.4 Simplifying Polynomials ...................................... 575
    17.4.1 Polynomial Programs 580
18. Calculus .......................................................... 587
  18.1 Differentiation ............................................... 587
    18.1.1 Calculus with Trigonometric Functions 589
      18.1.1.1 User-Defined Derivatives 590
    18.1.2 Formal Derivatives 592
  18.2 Taylor's Polynomials .......................................... 594
  18.3 Summations ..................................................... 595
    18.3.1 Summation Patterns 596
  18.4 Integration .................................................... 598
    18.4.1 Symbolic Integration 598
      18.4.1.1 Integration Patterns 601
      18.4.1.2 Derivative and Integral 602
      18.4.1.3 Adding Integration Patterns 604
    18.4.2 Numerical Integration 608
    18.4.3 Integration Strategies 611
18.4.4 Programs as Integrands 615
18.4.5 Multiple Integrals 616
18.4.6 Polynomial Approximations 617

19. The Time System ............................. 621
19.1 The Clock .................................. 621
  19.1.1 Setting and Reading the Time 622
  19.1.2 Time Arithmetic 625
  19.1.3 Date Arithmetic 626
19.2 The Alarm System ............................ 627
  19.2.1 Setting Alarms Manually 628
    19.2.1.1 Cancelling Alarm Repeats 630
  19.2.2 Appointment Alarms 630
    19.2.2.1 Unacknowledged Repeating Alarms 635
  19.2.3 Control Alarms 635
  19.2.4 Alarm Commands 636
  19.2.5 The Alarm Catalog 638
  19.2.6 Automatic Alarm Deletion 640

20. Statistics .................................. 641
20.1 Data Entry ................................. 641
20.2 One-Variable Sample Statistics .......... 644
20.3 Two-Variable Statistics .................... 646
  20.3.1 Correlations 648
  20.3.2 Regressions 649
    20.3.2.1 Pseudo-Linear Regressions 650
  20.3.3 Best Fit 653
  20.3.4 Scatter Plots with Error Bars 653
  20.3.5 Summary Statistics 655
20.4 General Least-Squares Fitting ............. 656
  20.4.1 Utilities 660
  20.4.2 Polynomial Fits 662
  20.4.3 Non-Linear Least Squares Fits 664
20.5 Probability Commands ....................... 669
  20.5.1 Combinations and Permutations 669
    20.5.1.1 Factorial and Gamma Function 671
  20.5.2 Random Numbers 671
20.6 Upper-Tail Probability Distributions ...... 673
  20.6.1 UTPN 674
  20.6.2 UTPT 674
  20.6.3 UTPC 675
  20.6.4 UTPF 675
  20.6.5 Probability Density Functions 677
13. Introduction to Part II.

The HP48 scientific calculator is renowned for its remarkable concentration of mathematical computation features in a small, highly portable package. Even a quick glance at the keyboard suggests its power and range—everything from the sines and cosines that are traditional for scientific calculators to more exotic features associated with customization and communications. The HP48 represents the culmination of eighteen years of calculator development at Hewlett-Packard, incorporating virtually every useful feature of its predecessors, plus many that have never appeared on any other handheld calculator. The symbolic algebra system incorporated in the HP 48 and its immediate forerunner the HP 28 is actually the most widely used computer algebra system in the world.

By HP48 we are referring to the HP48SX, which was introduced in 1990, and the HP48S, which followed in 1991. The lower-cost HP 48S shares an identical feature set with the HP 48SX except that it lacks plug-in memory ports. Throughout this book, it will generally not be necessary to distinguish between the two models, since all of the operations discussed here, and all of the example programs, execute identically on both types.

The HP 48 contains several hundred programmable commands that form the basis of its programmability and customizability. You can supplement the built-in command set with libraries, which are software packages designed to integrate with the permanent ROM-based system, to provide additional programmable and non-programmable features. Furthermore, you can use the built-in and library commands to create an endless variety of programs that make the HP 48 a very personalized tool. The straightforward and consistent design principles of the calculator ensure that this considerable array of programmed and preprogrammed operations is manageable in a straightforward and simple-to-use manner. These principles and methods are the subject matter of HP 48 Insights I: Principles and Programming.

It is possible to accomplish most tasks for which the HP 48 is suited by using commands or combinations of commands—programs. However, the HP 48 goes well beyond providing a simple list of commands, to provide a higher level of convenience and power for certain common computation tasks. In particular, for five mathematical topics plus personal time management, the HP 48 offers special menus that provide prompting and special operations to facilitate manual interaction, in addition to programmable commands. These six subjects are labeled by the single orange titles that appear above the \[7, 8, 9, 4, 5,\] and \[6\] keys: SOLVE, PLOT, ALGEBRA, TIME, STAT, and UNITS. The interactive features associated with the ALGEBRA are actually accessed via
13.0 Introduction

the EquationWriter (➡ EQUATION), but with that exception the menus are self-contained and focused on their respective subject areas.

Topics associated with these six menus form the principal subject matter of this book. The HP 48 Owner's Manuals refers to the menus as "applications." We prefer to use that term for the uses of the calculator and its features rather than for the features themselves. The subtitle Problem Solving Resources is more descriptive of the nature and intent of the six menus—the six are at least prime resources for tasks related to the six areas, as part of the full problem-solving arsenal of the HP 48.

13.1 About This Book

HP48 Insights II: Problem Solving Resources is the second volume of the two-part Insights series for the HP 48 calculator. For the sake of the master index provided at the end of this book, the page and section numbering is made continuous with that of HP48 Insights I. Thus the first chapter of this book is Chapter 13 (apologies to triskadekaphobes), and the first page is page 401. (The last numbered page in part I is 381, but starting part II at 401 makes it easier to remember for index page references, i.e. pages numbered higher than 400 are in part II).

To a considerable degree, this book stands alone from HP48 Insights I. Assuming that you have reviewed at least as much of the HP48 Owner's Manuals as is necessary to understand HP48 rudiments such as keyboard and menu operations, object entry, etc., you should be able to read the material and exercise the examples here without reference to part I. There are, of course, numerous cross references between the two parts, including a few uses in part II of programs that are listed in part I, but for the most part these are not crucial to the material in part II. For the sake of readers who have not read part I, at the end of this chapter we reproduce the descriptions of notation conventions and important terminology from Chapter I that are used throughout this book.

The principal subjects of HP48 Insights II are those associated with the six special menu keys mentioned in the previous section: numerical equation solving, plotting, symbolic algebra and calculus, the HP 48 time system, statistics and curve fitting, and physical unit management. There is no necessary order for presentation of these topics, so we will just follow the order in which the menus appear on the keyboard. You can actually study the chapters in any order; cross-references between the chapters will help you find other material related to any section you happen to be reading.

The following summarizes the chapter topics:
### 13.1 Topics

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>Introduction</td>
</tr>
<tr>
<td>14.</td>
<td>HP Solve</td>
</tr>
<tr>
<td>15.</td>
<td>Plotting</td>
</tr>
<tr>
<td>16.</td>
<td>Symbolic Objects and Solutions</td>
</tr>
<tr>
<td>17.</td>
<td>Symbolic Manipulations</td>
</tr>
<tr>
<td>18.</td>
<td>Calculus</td>
</tr>
<tr>
<td>19.</td>
<td>The Time System</td>
</tr>
<tr>
<td>20.</td>
<td>Statistics</td>
</tr>
<tr>
<td>21.</td>
<td>Unit Management</td>
</tr>
</tbody>
</table>

### 13.2 Notation

In order to help you recognize various calculator commands, keystroke sequences, and results, we use throughout this book certain notation conventions:

- All calculator commands and displayed results that appear in the text are printed in helvetica characters, e.g. DUP 1 2 SWAP. When you see characters like these, you are to understand that they represent specific HP 48 operations rather than any ordinary English-language meanings.
- Italics used within calculator operations sequences indicate varying inputs or results. For example, 123 'REG' STO means that 123 is stored in the specific variable REG, whereas 123 'name' STO indicates that the 123 is stored in a variable for which you may choose any name you want. Similarly, << program >> indicates an unspecified program object; { numbers } might represent a list object containing numbers as its elements.

- HP 48 keys are displayed in helvetica characters surrounded by rectangular boxes, e.g. [ENTER], [EVAL], or [EEX]. The back-arrow key looks like this: \( \leftarrow \), and the cursor keys like these: \( \langle \), \( \rangle \), \( \uparrow \), and \( \downarrow \).

- A shifted key is shown with the key name in a box preceded by a left- or right-shift key picture, \( \sqcap \) or \( \diamondsuit \), e.g. \( \sqcap \text{TIME} \), or \( \diamondsuit \text{PURGE} \). A shifted key is identified by the orange or blue label above the key, rather than the label on the key itself-\( \sqcap \text{SOLVE} \) rather than \( \sqcap \text{7} \).

- Menu keys for operations available in the various menus are printed with the key labels surrounded by boxes drawn to suggest the reverse characters you see in the display, like these: \( \text{SIGN} \) or \( \text{LIST} \).

Examples of HP 48 operations take several forms. When appropriate, we will give step-by-step instructions that include specific keystrokes and show the relevant levels of the stack, with comments, as in the following sample:

<table>
<thead>
<tr>
<th>Keystrokes:</th>
<th>Results:</th>
<th>Comments:</th>
</tr>
</thead>
<tbody>
<tr>
<td>123 [ENTER] 456 +</td>
<td>1: 579</td>
<td>Adding 123 and 456 returns 579 to level 1.</td>
</tr>
</tbody>
</table>

For better legibility, we don't show individual letters and digits in key boxes--we just show 123 rather than \[1\] \[2\] \[3\], and ABC rather than \(\alpha\) \(\alpha\) \(A\) \(B\) \(C\). Key boxes are used for multi-letter keys on the keyboard and in menus.

In some cases, a printed listing of the stack contents isn't adequate, so we use an actual HP 48-generated picture of the calculator display, such as this picture from Chapter 6:
A large number of the examples, however, are given in a more compact format than the keystroke example shown above. These examples consist of a sequence of HP 48 commands and data that you are to execute, together with the stack objects that result from the execution. The “right hand” symbol \(\Rightarrow\) is used as a shorthand for “the HP 48 returns...” In the compact format, the addition example is written as

\[
123 \ 456 \ + \ \Rightarrow \ 579
\]

The \(\Rightarrow\) means “enter the objects and commands on the left, in left-to-right order, and the HP 48 will give back--return--the objects on the right.” If there are multiple results, they are listed to the right of the \(\Rightarrow\) in the order in which they are returned. For example,

\[
\text{A B C ROT SWAP } \Rightarrow \text{ B A C}
\]

indicates that B is returned to level 3, A to level 2, and C to level 1.

Because of the flexibility of the HP 48, there are usually several ways you can accomplish any given sequence, so we often don’t specify precise keystrokes unless there are non-programmable operations in the sequence. If there are no key boxes in the left-side sequence, you can always obtain the right-side results by typing the left side as text into the command line, then pressing [ENTER] when you get to the \(\Rightarrow\) symbol.

The \(\Rightarrow\) symbol is also used in the stack diagrams that are part of most program listings. The stack diagrams show how to set up stack objects for execution of the program, where the objects to the left of the \(\Rightarrow\) are the “input” objects, and the objects following the \(\Rightarrow\) are the program outputs.

The most elaborate “examples” in this book are programs. Each program is listed in a box that includes a suggested program variable name, a stack diagram, the actual steps that make up the program, and comments to help you understand the steps.
following sample listing illustrates the various features of the format:

<table>
<thead>
<tr>
<th>SAMPLE</th>
<th>Sample Program Listing</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>level 3</td>
<td>level 2</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>[matrix]</td>
<td>n</td>
</tr>
</tbody>
</table>

```
<< A B → a b
<<
  IF C D
  THEN 1 2 → n m
  <<
  START E F
  DO G UNTIL H END
  NEXT
  >>
ELSE 1 J
END
>>
>>
```

1. The name of the program (SAMPLE) is listed first, followed by an expanded version of the name that is descriptive of its purpose. When you have entered the listed program, you should store it in a variable with the specified name. If no name is given, the program is just intended to illustrate some point in the text, and there's no need to give it any particular name.

2. The program’s checksum is listed at the end of the name line, as a four-digit hexadecimal number. If you enter the program into your HP 48, you can verify that you have entered it correctly by comparing the listed checksum with the value returned by BYTES (section 12.5.1) for your program.

3. Below the program name is a stack diagram, that specifies the program’s input and output on the stack. The program arguments are shown to the left of the →, and the results to the right. In the example, the stack diagram indicates that the program requires a string in level 3, a matrix in level 2, and a real number n in level 1, and returns a new matrix in level 1. The object symbols in the stack diagram are as descriptive as possible, showing not only the required object type but also the conceptual purpose of the objects. A stack diagram

```
length width height → volume
```

shows that a program takes three real numbers (no object delimiters) representing length, width, and height, and returns another real number that is the volume.
4. The program listing is broken into lines, where each line has one or more program objects listed at the left, and explanatory comments on the right. There may be just one object on a line, or several whenever the collective effect of the objects is easy to follow. You do not have to use the same line breaks (or any at all) when you enter the program.

5. Lists, embedded programs, and program structures start on a new line unless they are short enough to fit entirely on one line. More frequently, each program or list delimiter or structure word starts a new line. The sequences between the structure words are indented, so that the structure words stand out. In the case of nested structures, each structure word of a particular structure is lined up vertically at the same indentation from the left margin. (The structure word \( \rightarrow \) does not start a new line, but the local variable defining procedure that follows the \( \rightarrow \) does start a new line.) Note that when you edit a program on the HP 48, the program display follows these same conventions, within the limitation of the 22-character display or printer width.

6. The comments at the right of the listing describe the purpose or results of the program lines at the left. If you are creating a program using a personal computer text editor, you can include similar comments in your program, setting them off from the program objects using the \(@\) delimiter (section 6.4.3.1). An especially useful "comment" is a description of the contents of the stack that are obtained after the execution of a program line. In our listings, the stack contents are distinguished from ordinary comments by enclosing the stack objects between \(| |\) symbols. The leftmost object in the series is in the highest stack level; the rightmost is in level 1. Thus

\[
| a \ b \ c \ d |
\]

indicates that the object \( a \) is in level 4, \( b \) in level 3, \( c \) in level 2, and \( d \) in level 1.

We recommend that you use similar conventions when developing and recording your own programs. Whether you write programs out by hand and type them into the HP 48, or use a personal computer to write programs and transfer them to the HP 48 via the serial port, program stack diagrams and comments are invaluable for later understanding and modification of the programs. Of course, there will be many occasions when you create a program directly in the HP 48 command line without benefit of any program listing. In these cases, we still recommend that you afterwards make a listing, or copy the program to a personal computer file, so that you can recover the program if you lose it for any reason.
13.3 Terminology

Finding useful terminology to describe a computer system like the HP 48 with new or unusual features can be a substantial problem. We have to use existing English words that are close to the meaning we wish to convey, but the dictionary definitions of the words usually differ from their meanings as applied to the HP 48. Consider the word object: for the HP 48, object means any of the mathematical or logical elements that constitute the data and procedural building blocks of the RPL language, but you won't find that meaning in a dictionary (although it is close to the definition used in mathematics).

Our solution to this difficulty is to provide precise definitions of any terms that we use that are specific to the HP 48, and then use those definitions consistently throughout. In some cases, the definitions we offer may differ from those used in the HP 48 manuals, usually because we need more careful definitions to get across a particular point. For example, the owners' manuals do not make a distinction between execute and evaluate. We find that such a distinction is useful (section 3.3) because it simplifies the descriptions of related subjects, such as the nature of global name objects (section 3.6.1).

Two other important terms that arise frequently are mode and environment. A mode is a calculator setting, often associated with one or more flags (section 7.1), that determines how a particular keystroke or command will behave. For example, in polar mode, complex numbers and vectors are displayed in polar coordinates rather than the usual rectangular coordinates. An environment is a glorified mode, which determines the entire calculator interface, including the display, key actions, and available operations.

The “home base” for the HP 48 is the standard environment. In this environment, the display shows the status area, stack, and menu key labels. All keys are active, with their ordinary labeled definitions. If you press [GRAPH], the HP 48 switches to the plot environment. Here the display is devoted to a graph or other picture, the menu keys are restricted to a menu of plotting operations, and the remaining keys are either assigned additional plot actions or are inactive altogether. Pressing [ATTN] returns to the standard environment. Other environments include the EquationWriter, the MatrixWriter, and the equation and statistics matrix catalogs.

While introducing and using this kind of specialized terminology, at the same time we will be using an informal style that takes some liberties with the language to avoid unnecessarily stilted descriptions. “You are in the program branch menu” is almost a non-sequitor when taken out of context, but it reads more easily than “the current HP 48 menu is the program branch menu,” and its meaning is clear.
14. HP Solve

HP Solve is an interactive equation-solving system that has become an important feature on several advanced HP calculators, starting with the HP18C in 1987. This system was developed as a generalization of the “time-value-of-money” (TVM) key system that originated with the HP80 calculator. In HP Solve, variable menu keys replace the fixed TVM keys, and a general-purpose numerical root-finder replaces the TVM equation-solver. A root-finder first appeared as a calculator function on the HP 34C. That original algorithm has been refined several times in later calculators; the HP 48 is the first calculator to permit its application to equations containing physical units.

In brief, HP Solve combines a menu key interface with an iterative root-finder, for the numerical solution of any problem that can be expressed as a single equation containing one unknown variable. This powerful system relieves you of the need to rearrange or solve an equation by hand before entering it into the calculator, or to write programs to find solutions.

A “solution” to such a problem is the determination of a value of the unknown variable, called a root, for which the equation is satisfied—the left side equals the right side. The root-finder is a built-in program that finds a root of an equation by an iterative process. It starts with one or two estimates of the root that you supply, then adjusts the value of the unknown variable repeatedly to minimize the difference between the two sides of the equation.

The TVM system in HP calculators uses a highly refined solving algorithm that is customized for the time-value-of-money equation

\[ PV + (1 + I \cdot p) \cdot PMT \cdot \left( \frac{1 - (1 + I)^{-N}}{I} \right) = -FV \cdot (1 + I)^{-N} \]

where

- \( PV \) is the present value;
- \( FV \) is the future value;
- \( I \) is the periodic interest rate;
- \( N \) is the number of compounding periods;
- \( PMT \) is the periodic payment; and

\( p \) has the value 1 for “Begin” mode and 0 for “End” mode.
This equation can serve as an initial illustration of the HP 48 implementation of HP Solve. Using the EquationWriter, you can enter the equation as it is shown above; or, you can use the command line to enter it in a linear format:

\[
'PV + (1 + I * p) * PMT * ((1 - (1 + I)^N) / l) = -FV(1 + I)^N
\]

With the equation in level 1, press \( \text{SOLVE} \rightarrow \text{STEQ} \rightarrow \text{SOLVR} \):

The SOLVR key does two things:

- It displays the current equation, an algebraic object or program that is selected for solving, as specified by the contents of the variable EQ. In this case, the TVM equation was stored directly in EQ by \( \text{STEQ} \).

- It activates the HP Solve variables menu, or solver menu for short, which is derived from the current equation. This menu contains a menu key labeled with a variable name for each variable in the current equation.

The solver menu key labels are displayed in dark characters on a light background rather the usual light-on-dark, to indicate that the menu keys have definitions that are effectively reversed from those of normal menu keys. That is, in the VAR or CST menus, pressing an unshifted menu key labeled by a global name executes the name object. For data class objects (section 3.4), this is equivalent to recalling the object stored in the corresponding variable. A left-shifted menu key stores a stack object into the named variable. In the solver menu, the primary and left-shift definitions are exchanged: a primary menu key stores a stack object, and the left-shifted menu key returns an object to the stack. In this case, the returned object is not the object currently stored in the variable, but instead is a number that is computed to be a solution of the current equation. This means that for this value of the specified variable, the current equation evaluates to zero or near zero, given the stored values of all of the other variables in the equation.
For example, use the TVM equation to compute the monthly payment on a 30-year loan of $50,000, at an annual interest rate of 9%. Enter the values as follows:

- Set display to two decimal places.
- Principle value of loan.
- Monthly interest rate.
- Number of payment periods.
- End mode.
- Future value (loan paid off).

Now use the REVIEW key to check the entries:

At this point, you have not specified a value for the payment, so it appears with the value undefined. For this problem, you don’t need to specify any value (the calculator will start with the value zero), so you can proceed to solve for the payment. In general, even the unknown variable needs an initial value to guide the root-finder (see section 14.5). Press ➩ PMT:

The TVM equation uses the convention that a payment (out) is a negative amount, so the result is negative, returned as the tagged object PMT: −402.31. The status area of
the display also shows the result, plus a *qualifying message*, in this case *Zero*, that provides information on the nature of the solution (see section 14.4.1).

Underlying HP Solve is a numerical root-finder that iteratively adjusts the value of the unknown variable until the equation is satisfied to within the numerical precision limits of the calculator. Although the HP 48 root-finder is uncommonly sophisticated and robust, the unique power of HP Solve arises from its ability to analyze an equation automatically to create the solver menu, relieving you of any need to program beyond entering the equation as an HP 48 object. The menu is a great convenience, since it lets you enter and keep track of the variables' values, and solve for any of the variables. In particular, you can solve for one variable, then enter a new value for that variable and re-solve for another variable. In the current example, suppose that you can only afford a monthly payment of $300. Then you can solve to see how large a loan you can obtain:

\[-300 \quad \text{PMT} \quad \leftarrow \quad \text{PV} \quad \rightarrow \quad \text{PV}: 37284.56\]

Some other HP calculators that provide HP Solve do not require a shift key press to identify a variable for solving. Instead, solving occurs when you press two consecutive menu keys. This shortcut assumes that normally you will solve for a variable immediately after storing a new values for one or more of the others, which is a very natural way to use HP Solve. However, if you interrupt this flow by performing any non-store operation, you must then do an extra store with a menu key. The HP 48 instead uses the less ambiguous shift-to-solve method in order to preserve more flexibility in HP Solve calculations. For example, you can enter or compute values for several of the known variables together, then store them into the variables by pressing their respective solver menu keys. You could not do this if pressing two consecutive menu keys started the root-finder.

### 14.1 The Equation Entry Menu

Interactive use of HP Solve is centered on the HP Solve equation entry menu. When you press \( \text{\textasciicircum}\) to activate this menu, you see a display like this:

```
Current equation:
EQ: 'PV+(1+I*p)*PMT*<...'
4: 3: 2: 1: SOLVR ROOT NEW EDEQ STEQ CAT
```
The status area displays the current equation, preceded by the name of the variable in which it is stored. If there is no variable EQ in the current directory, the menu display looks like this:

```
No current equation.
Enter eqn, press NEW
```

The message provides some guidance for new users on how to create a current equation. The menu contains the following menu keys:

- **SOLVR** for activating the solver menu.
- **ROOT** for entering ROOT, the command form of the root-finder.
- **NEW** for entering and naming new equations.
- **EDEQ** to copy the current equation to the command line for modification.
- **STEQ** for storing an object directly from the stack into the variable EQ. \( \rightarrow \) **STEQ** recalls the current contents of EQ to level 1.
- **CAT** for reviewing and selecting potential current equations from memory.

**NEW** provides a convenient shortcut for activating a new HP Solve equation. After you have entered a new object into level 1, pressing **NEW** produces the following display:

```
Name the equation,
press ENTER
```
You now type a name, and press [ENTER]. Notice that the α annunciator is on, and you may type alpha-keys directly. [ENTER] stores the level 1 object in a global variable with the name you supply, and also stores that name in EQ so that the new object becomes the current equation.

When the level 1 object is anything other than an algebraic object, the initial [NEW] display looks like this:

```
α
RAD PRG
{ HOME }
Name the equation, press ENTER
.EQ
```

The .EQ (or ,EQ when flag −51 is set) is a suffix for the variable name that ensures that the variable will be included in the equation catalog (section 14.2.3.2). If you don’t want to use the suffix, press [ATTN] to clear the command line before you enter the variable name.

When the current equation object is a program or algebraic object of substantial size, the time it takes to show the object every time you press [SOLVE] or [SOLVE] can be a nuisance. You can suppress the automatic displays associated with these and all other HP 48 menus by setting flag −58. With the flag set, you can still produce the special displays on demand by pressing [REVIEW].

### 14.2 Basic HP Solve Operation

The general procedure for using the HP Solve is as follows:

1. Enter or compute the algebraic object or program representing a problem. Following the terminology of the HP 48 Owner’s Manual, we call this object the current equation, even when it is not literally an equation.

2. Store the problem object in a global variable, and store the variable’s name in a global variable named EQ. The variable EQ in the current directory is always used by HP Solve to identify the current equation. You can also store the current equation object itself directly in EQ. This is most suitable when you have no further use for the equation after solving it, since it is overwritten when you next choose a new current equation. The NEW operation (in the [SOLVE] menu)
provides a shortcut method of storing and selecting a new object.

3. Activate the solver menu. You can use \texttt{SOLVR} in the \texttt{[\downarrow] Solve} menu, or in the equation catalog menu, or \texttt{\rightarrow Solve} when any other menu is active. The solver menu contains a menu key for each independent variable in the current equation.

4. Enter values for each of the "known" variables--the variables for which you already know the values. Do this by entering a value into level 1, then pressing the appropriate menu key.

5. Store a guess for the solution value of the unknown variable, again by entering a value and pressing the variable's menu key.

6. Solve for the unknown: press \texttt{[\downarrow]}, then the menu key for the unknown variable. This starts the root-finder. When the root-finder is finished, it returns the solved value for the unknown to level 1, tagged with the variable's name, and stores that (untagged) value in the unknown variable. You will also see a qualifying message in the display that can help you interpret the result.

7. Verify the result, using the evaluation key \texttt{EXPR=} that appears at the end of the solver menu.

8. Repeat steps 2 through 6 with new values for the variables, and perhaps a new choice of the unknown variable.

A nice property of HP 48 Solve is that the current "equation" does not literally have to be an equation. You can use programs, expressions, and equations almost interchangeably for solving purposes. In effect, HP Solve always solves $f(x) = 0$. In the case of an HP 48 algebraic expression object, $f(x)$ is the expression represented by the object, where $x$ is the name of the unknown variable. For an equation object representing $g(x) = h(x)$, HP Solve solves $f(x) = g(x) - h(x) = 0$. For a program, $f(x)$ is the expression that is equivalent to that (RPN) program. Programs used for this purpose must be equivalent to an algebraic expression, taking no arguments from the stack and returning one real number or unit object.

\section{14.2.1 Independent, Dependent, and Unknown Variables}
Consider applying HP Solve to an equation \textquote{A+B=C+D}. If any of the four variables takes only numeric values, we call it an independent variable, because you can choose any values for it without regard to the other three variables. You can also choose any independent variable to be the unknown variable, and solve for its value rather than assigning it. The unknown variable is considered independent because you can at any point assign it a value and solve for one of the other variables.

Suppose now that you store the expression \textquote{B+C+F} as the value of the variable A.
This changes the role of $A$ to that of a dependent variable—you can no longer set its value arbitrarily, but must compute it from other variables. By storing an expression in $A$, you are saying that the symbol $A$ now is just an abbreviation for the expression.

Of all the variable names that appear in the current equation, only those that are independent variables appear in the solver menu. To see this, enter '$A+B=C+D$' as the current equation:

```
'A+B=C+D' \[<1]\[SOLVE] =STEQ Z=SOLVR:Z=:
```

Here you see menu keys for $A$, $B$, $C$, and $D$, assuming that these variables do not have procedures already stored in them. This indicates that they are all independent variables. Now execute

```
'B+C+F' [ENTER] EAZ,
```

then rebuild the solver menu by pressing $\Rightarrow$ SOLVE:

```
EQ: 'A+B=C+D'

4:
3:
2:
1:
A B C D Expr=

B C F D Expr=

```

- 416 -
Notice that F now appears in the menu instead of A. A is no longer independent, since it is defined in terms of B, C, and F, so it has been removed from the menu. You are really trying to solve ‘B+C+F+B=C+D’; the name A now is effectively an abbreviation for the expression ‘B+C+F’.

In some cases, you may wish to assign a value to a current equation variable and also remove the variable from the menu. For example, a physical constant like the speed of light might appear in the equation as a symbol, but there is no reason to include the symbol (name) in the solver menu, since you won’t need to change the value. In such cases, instead of storing the value itself in the variable, store a program consisting only of the value, e.g. << 2.99E10 >> for the speed of light. The equation will evaluate properly for HP Solve, but the solver menu will not include the variable. If you store the program by means of the variable’s solver menu key, then press → SOLVE to rebuild the solver menu, and the menu key will disappear.

In the context of the mathematical function plotting performed by DRAW, the term “independent variable” is used to refer to the variable corresponding to the horizontal axis (section 15.7.5). That is, the vertical coordinate represents the value of the current equation as a function of the independent variable. The values of all of the other variables in the current equation are held constant during the plot.

When you use DRAW in conjunction with HP Solve, you should use INDEP (section 15.2.5) to select HP Solve unknown variable as the independent plot variable corresponding to the horizontal coordinate. Then the plot will represent the value of the current equation as a function of the unknown. In particular, the roots of an expression are shown by the intersections of the curve and the horizontal zero axis. For an equation, DRAW plots two curves, which intersect at the equation’s roots.

14.2.2 An Example
All of the steps involved in HP Solve are demonstrated in the following problem, which has appeared as a programming exercise in several HP calculator manuals.

Example. Solve for t in the equation

\[ h = 5000(1 - e^{-t/20}) - 200t = 0. \]

(This is imagined as the equation of motion of a “ridget,” where h is the height in meters of the ridget above the ground, and t is the time since it was hurled into the air. Thus you are to solve for the time at which the ridget strikes the ground.)
Solution.

1. Start by entering the equation:

```
< MODES STD > 'H=5000*(1-EXP(-T/20))-200*T' ENTER
```

2. Name the equation, and select it as the current equation. Before doing this, however, you might want to save the TVM equation from the previous example:

```
'EQU' RCL 'TVM' STO
```

Now press `NEW`, then RIDGET ENTER. The new equation is stored in the variable RIDGET, and the name RIDGET is stored in EQ.

3. Activate the solver menu. You can do this by pressing `SOLVR` in the `< SOLVE` menu or in the equation catalog (section 14.2.3.2), or by pressing `SOLVE`. In this example, you see the following:

```
RIDGET: 'H=5000*(1-EXP
```

Notice the menu keys for the two variables in the problem, T and H, and the `EXPR=` key. The latter appears as the last entry in solver menus, and is used for verifying solutions (section 14.4). `EXPR=` is present on the second page of the TVM menu shown earlier, but you must press `NXT` to see it.

4. Enter values for the known variables. For this example, there is only one known variable, H, which is to have the value zero. Enter 0 `H`.

5. Enter an initial guess for the solution, which is necessary as a starting point for the root-finder's iteration. You can take your chances and enter any number that seems reasonable, but usually you can do better with a quick analysis. In this case, you can observe from the equation that
- There is a solution for $h = 0$ at $t = 0$; that solution is of no interest, so you should choose a positive guess away from zero.

- When $t$ is large, you can neglect the exponential term, approximating the equation as

$$5000 - 200t = 0,$$

which has the solution $t = 25$.

Since the exponential term is negative, the actual root must be less than 25. (The HP 41 Advantage Pac manual uses guesses of 5 and 6, which do return the right answer, but guesses closer to 25 and farther away from 0 are safer—that is, they are less likely to lead to the $t=0$ solution. Try a guess of 20, by entering $20 \hat{=} T \hat{=}$.)

Pressing $\leftarrow \text{REVIEW}$ shows the values of both variables:

![Screenshot of HP Solve showing variables and equation](image)

6. Solve for the unknown, by pressing $\leftarrow$ followed by the solver menu key for that variable. Here you are solving for $T$, so you press $\leftarrow \hat{=} T \hat{=}$ to start the root-finder. After a few seconds, the HP 48 returns the solution $9.28425508759$, plus the qualifying message Zero. The message indicates that the two sides of the equation are equal (difference zero) to 12 decimal places, when evaluated for this value of $T$. You can verify the accuracy of the result by pressing $\hat{=} \text{EXPR} =$ :
This shows that both sides of the equation evaluate to the same value, suggesting that the result is a valid root.

7. Vary the parameters, and re-solve the equation as desired. For example, you might like to know how high the ridge is at $T=5$:

\[ [H] = 105.99608464 \]

As long as you don’t make any extreme changes in any of the variables’ values after finding one solution, you should not have to enter new guesses for any variable you select as an unknown.

14.2.3 Equation Management
Although it is possible to use HP Solve entirely by storing one equation at a time in EQ, you can accumulate any number of equations stored in global variables, then pick and choose among them as needed. Several HP 48 features help you organize their storage and use.

14.2.3.1 Using Subdirectories
When working through any of the examples in the preceding sections, you may have noticed that an HP Solve equation’s variables appear in the VAR menu along with the equation itself and EQ. This is consistent with the usual HP 48 convention that storing into any variable is always done in the current directory (section 5.2). However, the convention is also that recall and evaluation of variables by name can take place from any subdirectory of the directory containing the variables. For HP Solve, this means that you can separate an equation from its variables by placing the variables in a subdirectory. There are several advantages:

- The directory containing the equation can remain relatively uncluttered.
- You can have multiple sets of stored values for an equation’s variables, storing each set in its own subdirectory.
- By using CLVAR, you can easily purge all of an equation’s variables at once, leaving the equation itself undisturbed.

To carry out an HP Solve exercise using a subdirectory, store the equation object in the parent directory and then switch to the subdirectory. The solver menu is built from the object specified by EQ in the current directory, so you must store the equation’s name in EQ in the subdirectory. Then activate the solver menu as usual, using $\mathbf{\text{ESOLVR}}$ or $\mathbf{\text{ESOLVE}}$. The menu works the same as it does when the equation object is in the
current directory, but as you store values using the menu keys, the variables are created in the current directory rather than in the parent directory with the equation.

You do not even need to keep all of the variables together. For example, when using the TVM equation from section 14.0, you might want to solve several different problems all using the same interest rate. In that case, you can store in the interest in a variable l in the same directory as the TVM equation, then create a new subdirectory for each problem. As long as you don’t store a new value for l in any of the subdirectories, the root-finder will use the value stored in the parent directory. (As described in section 14.2.1, you can even remove l from the solver menu by storing the program << rate >> in l instead of rate by itself.)

14.2.3.2 The Equation Catalog
As you accumulate stored equation objects for solving and plotting, the equation catalog is a valuable tool for finding and managing the equations. The catalog is activated by the CAT key in either the SOLVE or PLOT menus, or, from any other menu, by ALGEBRA. For example, assuming that you have created the TVM and the RIDGET variables from the preceding examples in the current directory, then CAT gives the following display:

```
{ HOME }
RIDGET: 'H=5000*(1-E...
TVM: 'PV+(1+I%p)*PMT...
```

The equation catalog is an environment (section 13.3), meaning that the display and keyboard are dedicated to its use. The primary feature of the display is a vertical listing of variables, each in the format name: content. Only the first eight or fewer characters of a name are shown, to leave room for displaying the stored object on the same line. The variables are shown in the same order as they appear in the VAR menu. Up to five variables are shown, along with a selection arrow—the triangle in the leftmost column. If the catalog lists more than five, you can move the arrow through the list with ▲ or ▼. △ ▲ and △ ▼ move the arrow up and down one “page” of five variables at a time. ▶ ▲ selects the first variable in the catalog, and ▶ ▼ selects the last.

The catalog does not show all of the variables in the current directory, but restricts its
listing to the following:

- Variables containing algebraic objects. These are presumed to be the most common entries for solving and plotting, so they are always included in the catalog.

- Variables with names ending in .EQ or ,EQ. This lets you include objects of any type in the catalog, especially programs and lists.

- EQ, regardless of its content. This reminds you of which object is the current equation.

- Directories. These are listed in the shortened format `name: dir`.

The catalog display can be slow when the stored objects are large, because of the time required to build the display form of each object. To move more quickly around the catalog, press `FAST` (in the second page of the catalog menu). This option suppresses the object display and shows only the variable names. The HP 48 remembers this mode between catalog sessions by means of flag -59, which is set for the fast display, and cleared for the full display. The white square in the key label indicates that the fast display is selected; pressing `FAST` again restores the full display.

You can move the catalog through the directory system without leaving the catalog environment. Moving the selection arrow to a directory, then pressing `ENTER`, switches to that directory and shows the catalog derived from its variables. `UP` and `HOME` are also active in the catalog environment, for moving upwards through the directory structure.

The selection arrow specifies a particular variable for the operations represented by the catalog menu keys. For example, moving the arrow to a particular variable and pressing `SOLVR` stores that variable's name in EQ and activates the solver menu. `PLOTR` similarly activates the `PLOT` menu. Other options:

- `VIEW` displays the selected object in its command-line form (up to four lines). The display persists while you hold the key down.

- `EDIT` copies the selected object to the command line, where you can edit it. As usual, `ENTER` stores the edited version back in the variable, and `ATTN` cancels the changes and returns to the catalog display.

- `-STK` copies the selected object to the stack, tagged with its variable name. For variables that are not directories, `ENTER` has the same effect as `-STK`

- `ORDER` (second page) moves the selected object to the beginning of the current directory, so that it appears as the first entry in the catalog. Pressing `ORDER` with a variable name selected is equivalent to executing `{ name } ORDER` (section 5.1).
• \texttt{PURGE} purges the selected variable from memory.

\texttt{ZEQ+} provides a means for applying certain catalog operations to a list of names. The first time you press \texttt{ZEQ+}, it creates a list containing the selected variable name. The list is displayed in the status area. Then you can move the selection arrow to other entries; each subsequent use of \texttt{ZEQ+} adds the selected name to the list. While such a list is displayed, pressing \texttt{SOLVR} stores the list in \texttt{EQ} and activates the solver menu; \texttt{PLOTR} works similarly. Also:

- \texttt{EDIT} copies the list to the command line for editing.
- \texttt{-STK} copies the list to the stack.
- \texttt{ENTER} copies the list to the stack, unless the selection arrow points to a directory. In that case, the selected directory becomes current and the pending list remains active.
- \texttt{ORDER} applies the command \texttt{ORDER} to the pending list, and clears the pending list.
- \texttt{ATTN} returns to normal catalog operation, discarding the list.

\texttt{VIEW} and \texttt{PURGE} apply as usual to the selected variable, ignoring any pending list.

14.3 Solving With Units

The current equation and any of its variables may include unit objects, and the result returned by HP Solve will be a unit object when appropriate. The general approach to solving with or without units is the same, but there are a few points to bear in mind.

- If the unknown variable requires units to satisfy the current equation, then the initial guess for the unknown must have units. The HP Solve result will be expressed in the same units as the original guess. It is not possible for the HP 48 to provide a default guess, nor can it determine even the dimensions of the result. (To do so would require a symbolic solution, which is not within the scope of HP Solve. See section 14.8.)

- You can use the solver menu to store unit objects as values for the variables. Once you have established the units of a variable by storing a unit object, you can change the magnitude of the unit object by entering a real number and pressing the menu key; this leaves the unit part unchanged. However, if you later want to use that variable in another HP Solve problem where it has no units, you can store a dimensionless number in the variable by enclosing the number in a list, or by purging the variable first.

- \textit{Example}. A railroad car weighing 5 tons rolls with an initial speed of 4 ft/s down a
225 foot ramp with a vertical drop of 3 feet, then across a level track 175 feet long, terminated by a spring-loaded bumper. The rolling friction of the car is 50 pounds force. What must the spring constant be in order to stop the car by compressing the spring 2 feet?

**Solution.** This example is most easily solved as a conservation of energy problem. That is, the initial kinetic plus potential energy of the car must equal the energy dissipated by friction plus the energy associated with compressing the spring by 2 feet. Expressing this as an equation, we have

\[ M \cdot \frac{V^2}{2} + M \cdot g \cdot H = F \cdot (R + L + X) + K \cdot X^2/2, \]

where \( M \) is the car mass, \( V \) is the initial speed, \( g \) is the acceleration of gravity, \( H \) is the ramp height, \( F \) is the frictional force, \( R \) is the length of the ramp, \( L \) is the length of the level track, \( X \) is the compression of the spring, and \( K \) is the (unknown) spring constant.

To use HP Solve, start by storing the above equation as the current equation:

\[ 'M*V^2/2+M*g*H=F*(R+L+X)+K*X^2/2' \]

The solver menu then looks like this:

```
RRCAR: 'M*V^2/2+M*g*H=F*(R+L+X)+K*X^2/2'  \{_SOLVE_}  \{NEW\}  RRCAR  \{ENTER\}
```

Since there are more than six variables in the equation, you must press \( \text{NXT} \) to see the remaining three variable keys plus \( \text{EXPR=} \).

Now enter the parameters from the problem:

- \( 5 \text{ ton} \) \( \Rightarrow M \)
- \( 4 \text{ ft/s} \) \( \Rightarrow V \)
Since the equation is linear in $K$, any initial guess will do, as long as it includes the appropriate units:

$$0 \text{lbf/ft} \equiv \frac{K}{6193.23} \equiv 6193.23 \text{lbf/ft}$$

### 14.3.1 Faster Solving with Units

Evaluation of algebraic objects containing units is intrinsically slower than ordinary numerical evaluation due to the unit conversions that take place as the evaluation proceeds. Since using a root-finder requires repeated evaluation of the current equation, solving an equation with units can sometimes take a long time. For faster solutions, you can of course remove all of the units from an equation and solve with dimensionless quantities, but then you have to do all of the unit conversions manually before and after solving. Short of this, you can often obtain substantial speed improvements by replacing the current equation with a modified version in which as many of the equation's calculations as possible have already been performed. (This will speed up solving with any equation, but the effects are most dramatic for equations containing units.)

The easiest way to preprocess an equation or expression to reduce its number of operations is to use `SHOW` (section 16.4.2). This command can selectively resolve some variables while leaving others unevaluated; for efficient solving the idea is to evaluate all names except that of the unknown variable. For example, after entering values for all of the variables in the railroad car problem, you can reduce the equation as follows:

```
RRCAR \{ K \} SHOW \equiv '16.2432380069 \text{ton*ga*ft}=20100 \text{lbf*ft}+(K*4 \text{ft}^2)/2'
```

Now only $K$ remains in the equation, and many of the arithmetic operations have been carried out. If you make this result the current equation, then solve for $K$, you will see that the solution is obtained much more rapidly. However, you can do even better by changing the equation into an expression, since one more unit subtraction is eliminated:

```
EQ\rightarrow \quad \text{COLCT} \equiv '12386.4760137 \text{lbf*ft}+(-2 \text{ft}^2)\text{*K}'
```
Using this expression as the current equation gives the fastest possible solution for K, using HP Solve or ROOT.

14.3.2 Monitoring Convergence
When a particular HP Solve solution appears to be taking a long time, you can elect to watch the root-search in progress, so that you can decide whether to continue or to terminate the process. Pressing any key while the root-finder is active creates a display like this:

```
-6193.23800692
+184.31199816
```

The status area shows two numbers, each preceded by a + or – sign; new values are displayed at each root-finder iteration. The numbers are the values of the unknown variable that bound the region in which the root-finder is searching; the + or – symbols indicate the sign of the current equation evaluated at the two points.

There are several circumstances that can cause the root-finder to iterate for a long time. For example:

- The current equation has the same sign at both edges of the region defined by the initial guesses. If the region includes a root, the root-finder can usually home in on it quickly. However, if there is no root in the region, the root-finder has to search outside of the region. The search may take many iterations before finding a sign change. In that case, when you monitor the search you will see the numbers changing while the two signs remain the same.

- The current equation is relatively “flat” around a root. In this case, the two signs are opposite, but it takes many iterations to narrow the search region to isolate the root. You will see the two numbers slowly get closer in value, with one or the other changing by a few decimal places at each iteration. The example ‘IFTE(X>0,1,-1)=0’ discussed in the next section is an extreme case; it can take several hundred iterations to refine the successive values to 0 and 1E–499.

- The root-finder is searching in a region where the current equation is constant.
14.4 Interpreting Results

Not all HP Solve problems work out as nicely as the ridget example. An equation may have multiple solutions, no solution, discontinuities, infinities, etc. It is important when you use HP Solve that you do not accept the “solution” at face value, but take some steps to interpret the result to assure yourself that it is a meaningful solution. This is the purpose of ZEXPR=Z and of the qualifying message.

The ZEXPR=Z key is provided in the solver menu as a ready means for evaluating the current equation to verify a solution:

- When the current equation is an expression or a program, EXPR= evaluates it, tagging the result with EXPR:. Following a successful HP Solve solution, the result should be zero or near-zero.

- When the current equation is an actual equation, EXPR= evaluates the two sides of the equation separately, returning two results tagged with LEFT: and RIGHT:. The equality or near-equality of these two results is a measure of the validity of the solution.

EXPR= is handy for "table-filling" evaluation with symbolic values as well as numerical. Consider, for example, filling in the following table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A+B+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.5</td>
<td>13.7</td>
<td>50.1</td>
<td>?</td>
</tr>
<tr>
<td>14.9</td>
<td>6.4</td>
<td>75.0</td>
<td>?</td>
</tr>
<tr>
<td>11.7</td>
<td>22.2</td>
<td>64.3</td>
<td>?</td>
</tr>
</tbody>
</table>

The task is to sum the each row of entries for A, B, and C. No root-finding is necessary, but the solver menu is still helpful. After storing 'A+B+C' as the current equation, each sum can be computed like this:

12.5 [A] 13.7 [B] 50.1 [C] [EXPR=] $\Rightarrow$ EXPR:76.3

You can also store an equation like 'SUM=A+B+C' in EQ, and use $\sum$ in the solver menu to find each sum, but using the root-finder is always slower than a straightforward evaluation with EXPR=.

In some cases, EXPR= may not return numerical results. It uses EVAL rather than -NUM, so that only one level of evaluation is performed (section 3.5.2). You can always
force a numerical result by setting flag -3 before using EXPR=, or by executing -NUM afterwards.

14.4.1 Qualifying Messages
The message Zero obtained in the ridget example is the most welcome of the possible qualifying messages. Zero means that the HP 48 has found a value of the unknown variable for which the current equation numerically evaluates to exactly zero. "Exactly" in this sense means exact to 12-digit precision, the numerical accuracy of the HP 48. When you see the Zero message, you are assured that ZEXPR= will return a single value of zero, or two equal values for the left and right sides of an equation.

When HP Solve returns the qualifying message Sign Reversal, it means that it is unable to find a value of the unknown that exactly satisfies the current equation. But it did find two values of the unknown that differ only in the twelfth digit, for which the corresponding values of the equation have opposite signs. This means that the equation crosses the zero axis somewhere between the two values, and so either value may be a good approximation of a solution. However, the calculator can’t tell for sure that there is a solution between the two values. For example, HP Solve returns the value 1.E-499 as the “solution” for the equation 'IFTE(X>0,1,-1)=0', which has a discontinuity at X=0. The Sign Reversal message warns you to check the solution.

The most immediate method of testing the result is to use ZEXPR= as described in the previous section. In the case of 'IFTE(X>0,1,-1)=0', ZEXPR= returns LEFT: 1 and RIGHT: 0. The disparity between these two results indicates that this may not be a proper solution. To check further, you should plot the current equation to get a visual indication of its behavior.

If you solve the equation 'X^2=2', you obtain the result 1.41421356237, with the Sign Reversal message. In this case pressing ZEXPR= returns LEFT: 1.99999999999 and RIGHT: 2.00000000000. The near-equality of these two values indicates that you have a good solution.

Other possible qualifying messages are as follows:

**Bad Guess(es)** The root-finder returns this error when it is unable to proceed because the guess or guesses that you supplied yield equation values that are not real numbers. For example, you will see this message if you solve '√X=2', and start with a guess of X= -5 (for which √X is imaginary).
Constant? This error occurs when the equation returns the same value for every value of the unknown tried by the root-finder. The equation is either a constant, or the guesses are in a region where the equation varies so slowly that the root-finder can't make any progress towards finding a root.

Extremum The root-finder has found a local minimum or maximum instead of a root. See section 14.7.

14.5 First Guesses

You may find it disappointing that a system as sophisticated as HP Solve requires you to supply a "first guess" of the answer to start the root-finder, so that it will return the "right" answer. After all, the TVM system on HP financial calculators, which is a specialized solver, doesn't require a first guess, yet it always returns the right result.

The need for an initial guess arises because generally equations can have more than one solution (or no solution at all). For example, \( x^2 + x - 12 = 0 \) has solutions at \( x = 3 \) and \( x = -4 \); the equation \( \cos(\sin x) = y \) has infinitely many solutions for \( x \). Furthermore:

- It is not practical for the calculator to attempt to determine how many solutions a particular equation has.
- Of all the possible solutions to an equation, there is no way for the calculator to know which one you want, based only on the equation itself.

The first point is true because the calculator can find solutions only by searching for them. Roots may occur anywhere between plus and minus infinity. To cover this range, of course, would require an infinite number of steps—or at least a very large number for the finite range of the calculator (from \(-10^{500}\) to \(10^{500}\)). This is obviously not practical. On the other hand, if HP Solve took relatively few steps to cover the real number domain, it could easily skip right over a region containing a root, and never find it. In short, an automatic solver can never know when it's finished finding roots, no matter how many it finds. The only reasonable thing for it to do is search until it finds one root, then quit.

The second point is a statement that a choice among multiple roots of an equation cannot be represented within the equation itself—it has to come from external information, often from some physical situation. For example, if the TVM equation somehow gave a negative solution for the interest rate, you could reject that solution as meaningless. But the information that the equation is only supposed to be valid for positive interest is not contained in the equation itself. As another example, consider the equation \((x-2)(x-3) = 0\), which has roots at \(x = 2\) and \(x = 3\). Which root is "correct?"
These considerations lead to the requirement that you must specify an initial guess for the root-finder. Your guess tells the root-finder where to start looking for a root, and guides it to a particular root among multiple roots.

14.5.1 How Many Guesses?
The HP48 root-finder begins execution by trying to find a region of values of the unknown variable in which the value of the current equation changes signs. That is, at the two boundaries of the region, evaluating the equation must give values with opposite signs. (The “sign” of an equation in this sense is the sign of the difference of the left and right sides.) Then it narrows the region until it contains just one point at which the equation is exactly satisfied (Zero message). Failing that, it finds two neighboring points where the equation has opposite signs (Sign Reversal). You will obtain the fastest results from HP Solve by specifying a good initial search region by means of the initial guess.

The HP 48 gives you the option of making single, double, or triple initial guesses for HP Solve. A single guess is a number, a double guess is a list of two numbers, and a triple guess is a list of three numbers. Any of the numbers can be complex--only the real parts are used (this is a convenience provided to let you use the coordinates of points digitized from plots, which are returned as complex number objects). The best choice is usually the double guess; it is more reliable than a single guess, and the extra certainty provided by the triple guess is seldom necessary. A good double guess contains two values of the independent variable that

- define a region in which the equation is well-behaved (no discontinuities, non-real values, or infinities) and which contains the root you want, and only that root; and

- yield values of the current equation with opposite signs.

With such a double guess, HP Solve will always home in quickly on the correct root.

It is often sufficient to supply only a single guess that is closer to your desired root than to any other root or extremum. HP Solve takes your single guess and makes its own second guess by duplicating the value and adding a small amount. Unless there actually is a root between the two guesses, the search starts to look outside of the initial region. Then, it is a matter of chance whether it finds the root you want, or some other root or extremum first. It is never guaranteed that the root-finder will find a particular root from a single guess, unless the equation has only one root (and no extrema). However, a single guess usually suffices in cases where you are using HP Solve repeatedly on the same equation, where each time you vary one or more of the independent variables by small amounts. The last solved value of the unknown variable is likely to be close enough to the new value to be a good single first guess. Since that value is already
stored in the unknown variable, it will be used as the first guess unless you explicitly replace it with another guess.

When you interrupt HP Solve by pressing [ATTN], it returns a list of three numbers. This list is a record of where the root-finder was searching when it was halted, and you can use it as a triple guess to tell the root-finder where to resume its iteration. If you want to restart the root-finder, store the list in the unknown variable, then solve for the unknown. You can also supply your own triple guess as the fastest way to make HP Solve find a particular root. Choose the first number in the triple guess list as your best estimate for the root, and choose the second and third numbers to bound the search region as in the double guess case.

Table 14.1 summarizes the meaning and application of the single, double, and triple guess options for HP Solve.

### Table 14.1. HP Solve Guesses

<table>
<thead>
<tr>
<th>Type of Guess</th>
<th>Meaning</th>
<th>When to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>One value close to a root.</td>
<td>To solve equations with only one root; to re-solve after adjusting the values of the independent variables.</td>
</tr>
<tr>
<td>Double</td>
<td>Two values on opposite sides of a root, where the values of EQ have opposite signs.</td>
<td>To guarantee that the root found will be the one between the two guesses, for equations with multiple roots and/or extrema.</td>
</tr>
<tr>
<td>Triple</td>
<td>First guess in the list is a best guess of the root; the 2nd and 3rd surround the root as in the 2-guess case.</td>
<td>To resume an interrupted root-search. Also is faster in general than the 2-guess case.</td>
</tr>
</tbody>
</table>

**14.5.2 Examples Using** $x(x-2)(x+2) = 0$

To illustrate the effect of different guesses, we will use various types of guesses in solving the equation $x(x-2)(x+2) = 0$ repeatedly. This cubic equation obviously has roots at $x = -2, x = +2,$ and $x = 0,$ so you know what to expect from HP Solve. To get an idea of how quickly the root-finder finds a root in each case, we will count the number of iterations the root-finder makes, by recording each execution of the current equation. This is achieved by the following:

```
'(X-2)*(X+2)*X'  'CUBEX' STO
```
When you solve for X by pressing <F9>, the value of N is incremented by 1 each time the latter program is executed. The difference in N before and after solving is the number of root-finder iterations. \( N \) appears in the solver menu, where you can use it to reset N to zero before each trial, but it has no effect on the root search. Here are the results of various trials:

<table>
<thead>
<tr>
<th>Initial Guess</th>
<th>Result</th>
<th>Iterations</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>-2</td>
<td>17</td>
<td>Found the most negative root.</td>
</tr>
<tr>
<td>+10</td>
<td>+2</td>
<td>17</td>
<td>Found the most positive root.</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td>11</td>
<td>Found the root closest to the guess.</td>
</tr>
<tr>
<td>1.1</td>
<td>0</td>
<td>13</td>
<td>Did not find the root closest to the guess.</td>
</tr>
<tr>
<td>{1.1 10}</td>
<td>2</td>
<td>14</td>
<td>Found the root between the two guesses. Fewer iterations than with the single guess 10.</td>
</tr>
<tr>
<td>{-9 3}</td>
<td>-2</td>
<td>9</td>
<td>Did not find the root between the two guesses--the equation has the same sign at both guesses.</td>
</tr>
<tr>
<td>{1.9 1.1 10}</td>
<td>2</td>
<td>8</td>
<td>Found the root between the two guesses; faster than the {1.1 10} case because of the additional “best guess.”</td>
</tr>
</tbody>
</table>

To specify multiple guesses for a problem involving units, you can use a list of two or more guesses as in the dimensionless case. One of the list elements should be a unit object with the correct units for that variable. If more than one element is a unit object, the units of the last element with units are used for the root-finder--the other elements are presumed to have the same units, and only their magnitudes are used. If you interrupt the root-finder with [ATTN], the first element of the three-element list returned will be a unit object.
14.6 Obtaining Guesses

In the preceding sections we discussed the need for initial guesses for the root-finder, without much explanation of how to obtain good guesses to supply. There are three general approaches you can use:

- Use the default guess.
- Use mathematical approximation.
- Use guesses obtained from a plot of the current equation. This is the easiest method, and generally the most reliable.

If you don't explicitly store a guess in the unknown variable before solving, HP Solve uses a default value to start the root-finder iteration. The default value is simply the current number stored in the unknown variable, or the number zero if the unknown variable has no current value. When you use the default guess, you are trusting that HP Solve will happen to find the correct root. This will certainly be the case if the equation has only one solution, and no extrema or other properties that might prevent the root-finder from converging on that value. If the current value of the unknown happens to be sufficiently close to a root, it is likely but not certain that the root-finder will return that root.

Using mathematical approximation to obtain a guess consists of studying the equation and trying to estimate the root from the mathematics of the equation. We did this in the problem of the ridget flinger in section 14.2.2. As another example, consider solving \( \cos x = x \). For small \( |x| \), \( \cos x \approx 1 - \frac{1}{2}x^2 \). Substituting this approximation in the equation, you obtain the quadratic equation \( 2 - x^2 = 2x \), which has the solutions \( x = \pm \sqrt{3} - 1 \). The choice of the negative root gives a value of \( x \) too large for the approximation to be valid, leaving the positive root \( x = \sqrt{3} - 1 \approx 0.732 \). With this value as a first guess for \( X \) for a current equation '\( \cos(X) = X \)', HP Solve returns the result \( 0.7391 \) for \( X \), which is quite close to the guess. The approximation method evidently can provide very good first guesses, but it does require some mathematical skill and intuition.

The preferred method of obtaining guesses is to plot the current equation and view the resulting curve to identify roots. Furthermore, you can invoke the root-finder directly while in the plot environment, using the plot cursor to specify an initial guess. Usually, you can place the cursor so close to a root that the single guess is sufficient to obtain the desired solution. You can use a plot to obtain single, double, or triple guesses and ensure that HP Solve finds the root you want. This application of plotting is an important reason for the existence of the plotting feature in the HP 48.

The key point to remember when you plot an expression is that the roots of the
expression are the values of the independent (abscissa) variable for which the plotted curve intersects the horizontal axis. Thus you can literally see the roots in the plot (assuming that the drawn horizontal axis passes through the origin, which is the usual case). For an equation, the HP 48 plots the left and right sides of the equation independently. The roots of the equation are the independent variable values for which the two sides have the same value. The roots of equations appear as the intersections of the two curves.

To illustrate the plot/solve process, consider the following problem.

**Example.** A rocket makes a round trip to α-Centauri, 4 light-years from the earth. On both legs of the voyage, the rocket accelerates uniformly to the half-way point, then turns around and decelerates at the same rate until it arrives at its destination. What acceleration is required so that the trip can be accomplished in 20 years of earth time? By uniform acceleration we mean that the astronauts experience a constant acceleration in their own frame of reference, so that they feel as if they were in a uniform gravitational field.

**Solution.** Since the trip divides into four kinematically identical 2 light-year segments, we only need to consider one segment, which must be traversed in 5 years. Looking at the first segment, let $X$ be the distance from the earth and $T$ the elapsed time, both measured on earth clocks. These are related by the formula

$$X = \frac{c^2}{A} \left[ \sqrt{1 + \left( \frac{AT}{c} \right)^2} - 1 \right].$$

where $A$ is the unknown acceleration, and $c$ is the speed of light. Enter this formula using the EquationWriter, or enter the linear form in the command line:

'\(X - c^2/A*(\sqrt{(1 + (A*T/c)^2)} - 1) + 0\_lyr\)'

and store it as the current equation:

- [SOLVE] [NEW] [SPACE] [ENTER] .

The 0_lyr at the end of the equation ensures that both sides of the equation have the same units. Without this, a plot of the equation would have a misleading appearance, since the units are dropped when plotting (see section 21.3.7).

Now use the solver menu ([SOLVR]) to enter the various parameters:
The last entry serves to establish the units for A, which are required to make a plot. Continue as follows:

\[
\text{PLOT EXYPEZ =SFUNC: E MODES: -30 SF}
\]

It is necessary to set flag $-30$ so that both sides of the equation will be plotted (see section 15.4).

\[
\text{PLOT } 'A' \text{ INDEP } .001 2 \text{ XRNG } 0 5 \text{ YRNG}
\]

The units of the vertical coordinate are determined by the dimensions of $X$. The right-hand side of the equation is the distance traveled in 5 years, which can not be greater than 5 light years, so we choose a range from 0 to 5. The horizontal plot limits are set from 0 to 2 (ga), which is a range of accelerations that might be tolerable to human beings--if there is no solution in this range, then the space trip is certainly impractical. Now make the plot by pressing [ERASE DRAW]:

Since the plotting is fairly slow, you might stop the plotting by pressing [ATTN] as soon as you see the curve representing the right-hand side intersect the horizontal line at $X=2$. To find the solution, move the cursor over to the vicinity of the intersection, and press [FCN ISECT], which returns the coordinates of the intersection. The computed acceleration, $A=0.185$ ga, is comparable to the acceleration of gravity on the surface of the moon.
Once you have used the interactive plot environment to find a solution for a problem, you can return to the solver menu to find additional solutions as you vary the problem's parameters. Generally, you won't change any of the parameters enough from trial to trial to require reploting for determining new first guesses. In the current example, you might ask how long the trip might take if the acceleration were increased to 1 \( ga \):

\[
\texttt{SOLVE} \quad 1 \quad \texttt{A} \quad \texttt{T} \quad \texttt{T: 2.81\_yr.}
\]

The round trip time is four times this result, or 11.2 years.

You can also use a plot to obtain guess values, without immediately using \texttt{ROOT} or \texttt{ISECT} to find a solution. As described in section 15.3.1, pressing \texttt{ENTER} returns the cursor coordinates to the stack as a complex number object. If the unknown variable is dimensionless, you can store the complex number directly in the variable as a guess--HP Solve uses only the real part. If the unknown requires units, you should not store the complex number. Use \texttt{RE}, or \texttt{[2D DROP]}, to extract the real part, and store that in the unknown.

### 14.7 Finding Critical Points

Although HP Solve is designed for finding roots of expressions, you can also use it to find critical points. These are points at which the derivative of a function is zero--that is, the critical points of \( f(x) \) are the roots of \( df/dx = 0 \). Therefore, if the current equation can be differentiated, the most straightforward method of finding its extrema is to differentiate it, make the derivative the current equation, and solve for the roots of the derivative. The roots of the derivative have the same abscissa values as the extrema of the original expression. Note that this method will also find inflection points, which are points at which the first and second derivatives are zero (at maxima, the second derivative is negative; at minima it is positive).

This process is automated by the \texttt{EXTR} operation, in the FCN sub-menu of the interactive plot menu. To illustrate the process, reactivate \texttt{CUBEX} (section 14.5.2) as the current equation, and plot it:

\[
\texttt{MODES} \quad 3 \quad \texttt{FIX} \quad \texttt{CUBEX} \quad \texttt{PLOT} \quad \texttt{STEQ} \quad \texttt{PLOT} \quad \texttt{NXT} \quad \texttt{RESET} \quad \texttt{PREV} \quad \texttt{X} \quad \texttt{PURGE} \quad \texttt{X} \quad \texttt{INDEP} \quad \texttt{ERASE} \quad \texttt{DRAW} :
\]
You can make better use of the plot screen by using the zoom operation:

\[
\begin{align*}
\text{ZOOM} & \quad X = .4 \quad \text{ENTER} \\
\text{ZOOM} & \quad Y = 1.5 \quad \text{ENTER} :
\end{align*}
\]

(You don't have to wait for the first zoom to finish replotting before starting the second. You can press [ATTN] immediately after the first [ENTER], then [ZOOM] again to enter the parameter for the second zoom.)

To determine the coordinates of the relative maximum, move the cursor to the left to near the \( x \)-position of the maximum, then press [FCN] [EXTR] :

\[
\text{EXTRM}: (-1.155, 3.079)
\]
Notice that the cursor moves to the maximum point; this will always happen unless the maximum is not visible. In that case, (OFF SCREEN) is displayed along with the qualifying message.

It is instructive to combine the plot of the current equation with that of its derivative. When the current equation is an expression or an equation, this is easy to achieve by using $\frac{d}{dx}$ (in the second page of the FCN sub-menu):

The first curve drawn is the derivative, which for this example is a quadratic. The second curve is the original (cubic) current equation. Notice that the extrema of the cubic occur at the same values of $x$ as the zeros of the quadratic.

When the current equation is not differentiable for any reason, you can still use HP Solve to find its extrema. Since the root-finder searches for a minimum of absolute value, it will stop when it encounters a local minimum that is above the horizontal axis, since movement in either direction produces an increase in the current equation value. A similar effect occurs for a local maximum that is below the axis. In either case, the value returned by HP Solve or ROOT reflects the position of an extremum rather than a root (HP Solve shows the qualifying message Extremum).

In the current example, the maximum is above the axis, and the minimum is below, so the root-finder will not find either extremum. However, you can still find the extrema without differentiating by adding a constant to the original equation such that a maximum is pushed below the axis, or a minimum is pushed above. For example, to find the maximum near $X=-1.15$, add $-4$ to the equation, then solve:

```
CUBEX 4 - \texttt{SOLVE} \ \texttt{NEW} \ \texttt{CUBEX2} \ \texttt{ENTER}
\texttt{SOLVR} -1.15 \texttt{X} \leftarrow \texttt{X} \ \rightleftharpoons X: -1.15469997367
```
14.8 Using ISOL with HP Solve

Since \texttt{EXPR=} is faster than numerical solving, it is often advantageous to attempt to solve the current equation symbolically as a preliminary to using HP Solve. This is particularly true in cases where you only intend to solve for one variable in the current equation. By using ISOL (section 16.4.1) once to solve the equation, you can reduce the problem from a root-finding task to one of straightforward evaluation using \texttt{EXPR=}.

Consider, for example, the simple travel cost problem:

\[ \text{COST} = \text{DIST} \times \text{PPG} / \text{MPG}, \]

where DIST is the distance traveled in miles, PPG is the price per gallon of gasoline, and MPG is the number of miles per gallon. Imagine that you want to construct a table of distance as a function of cost. If you enter the above equation as the current equation, you have to solve numerically for DIST after every new entry for COST. To save time, you can instead solve the equation for DIST symbolically. First, purge the variables, to ensure a fully symbolic solution:

\{ \text{COST DIST PPG MPG} \} \text{ PURGE.} \]

Then solve for DIST:

\[ \text{COST} = \text{DIST} \times \text{PPG} / \text{MPG} \quad \text{DIST} \quad \text{ISOL} \quad \Rightarrow \quad \text{DIST} = \text{COST} \times \text{MPG} / \text{PPG} \]

If you store the result as the current equation, you can then use the solver menu to enter the constant values for PPG, and MPG, and then successive values for COST. After each COST entry, pressing \texttt{EXPR=} returns LEFT: DIST, and RIGHT: value, where value is the distance computed for the latest COST value.

When you use \texttt{ISOL} as a preliminary to HP Solve, keep in mind that \texttt{ISOL} works only at the “top level” of an expression or equation. It does not evaluate any of the names that appear within the object, so that implied references to the unknown variable are not made explicit. For example, consider the equation \( X+Y=Z \), where Z has the value \( X+Y \). Solving the equation for \( X \) using \texttt{ISOL}, you obtain the result \( X=Z-Y \). But this result evaluates to \( X=X-Y+Y \), showing that the “solution” is meaningless. To guard against this type of problem, it is wise to use \texttt{SHOW} (section 16.4.2) before applying \texttt{ISOL} to make explicit all references to a specified name. In the example, \( X+Y=Z \) \( 'X' \) \texttt{SHOW} returns \( X+Y=X+Y \), making it obvious that the equation is trivial. \texttt{SHOW} is preferred over \texttt{EVAL} for this purpose because a) you only need to execute it once, whereas \texttt{EVAL} may have to be used repeatedly; and b) \texttt{SHOW} only evaluates names that reference the argument name at some level, keeping the result expression as compact as possible.
possible.

HP Solve on some other HP calculators automatically attempts an ISOL-like solution whenever you solve for a variable. In many cases this can speed up the solution by avoiding the use of the iterative root-finder, and in some cases may find a solution where the root-finder can not. This is not done on the HP 48 for several reasons:

- HP Solve is oriented toward finding a single solution, selected by you by means of an initial guess. A symbolic solution would have to be a principal value (section 16.5), which might prevent you from obtaining the particular root you want. Or it could be a general solution with arbitrary signs and integers, which would be inconvenient for repeated solving.

- A symbolic solution is only possible in certain problems, but the calculator would have to attempt it every time you solved, which could waste a lot of time. It would have to execute SHOW as well as ISOL, to insure that all references to the unknown variable are made explicit before solving.

- Since ISOL is an available command, you can choose whether to use it or HP Solve in a particular situation, rather than have the HP 48 impose its choice on you.

14.9 Programmable Solving

The root-finder used by HP Solve is available for program use as the command ROOT. This command requires as arguments the same elements that are needed for an HP Solve exercise:

- A program or an algebraic object (level 3) that acts as the current equation. The variable EQ is not used by ROOT.

- The name (level 2) of the unknown variable.

- A number or list of one, two, or three numbers (level 1), that acts as an initial guess for the root-finder.

All of the same considerations of the number and nature of the initial guesses discussed in section 14.5 apply to program root-finding as well. Of course, since a program can’t make visual inferences from a plot, it must include specific guesses, or logic to determine appropriate guesses.

ROOT returns its result for the unknown variable to the stack, and stores it in the variable. You can not monitor the root-finder during execution of ROOT as you can during an HP Solve exercise.

- Example. Find the positive roots of $\cos nx = e^x$, for $n=1,2,$ and 3.
Solution. Executing the program below returns these results:

0.601346767726 0.466819325092 0.369256413055

```
<< RCLF RAD -> f
<< 1 3
FOR n
 'COS(n*X) = EXP(X) - 1'
 'X'
 { 0 2 }
 ROOT
 NEXT
 f STOF
 >>
```

Set radians mode.

For n = 1, 2, 3:
Equation to solve.
Unknown variable.
Bracket the positive root.
Find the root.
Restore trigonometric mode.

The HP 48 plot below shows the exponential curve intersecting the three cosine curves. The current equation for the plot is the list (see section 15.2.2)

```
{ 'COS(X)' 'COS(2*X)' 'COS(3*X)' 'EXP(X) - 1' }.
```

14.10 Secondary Results

HP Solve is designed primarily to solve one equation for one unknown. However, the system does allow you to work with more than one equation, helping you to find a value for an unknown in one equation and use it as a known variable in the next (this process is automated further by the multiple equation solver in the HP 82211A Solve Equation Library Application Card). In place of the single object stored as the current equation, you may use a list of objects—any mixture of programs, algebraic objects, and the names of variables containing programs or algebraic objects.
For example, returning to the space voyage example of section 14.6, you might like to know how fast the rocket is traveling at the midpoint of the trip each way, and how much the astronauts age during the trip. Calling the astronauts' time (proper time) \( \tau \), and the speed \( v \), these quantities are given by

\[
\tau = \frac{c}{A} \arcsinh\left(\frac{A \cdot T}{c}\right)
\]

\[
v = c \tanh\left(\frac{A \cdot T}{c}\right)
\]

Because the HP-48 hyperbolic functions do not allow arguments with units, you must include a UBASE function in the functions' arguments in these equations. Enter the following:

\['SPACE' \{ '\tau = c/A \ast \text{ASINH(UBASE(A*T/c))}' 'V = c \ast \text{TANH(UBASE(A*T/c))}' \} \text{ STO+}
\]

This replaces the contents of SPACE with a list containing the original equation followed by the two new equations. The order of the equations in the list matches the order in which they are to be solved.

Now make SPACE the current equation, by entering 'SPACE' , or by using the catalog (section 14.2.3.2). Then \([\text{RPN}] \text{ SOLVE} \text{ STEQ} \) shows the following:

\[
\begin{align*}
X &= c^{2/A} \cdot (\sqrt{1+(A \cdot T/c} \\
X &= 2.1 \text{yr} \\
c &= 1 - c \\
A &= 1.8a \\
T &= 2.80623289867 \text{ yr}
\end{align*}
\]

(The values shown are those that would be left from the example in section 14.6. To work the examples below, you should re-enter these values for any variables that don't match these.) Notice that the menu now includes \([\text{RPN}] \text{ NXEQ} \) , which stands for "next equation." Pressing that key activates a menu for the proper time equation:
You can use the value of $T$ as a first guess for $\tau$:

$\tau = \frac{c}{A} \times \text{ASINH}(\text{UBASE})$

The astronauts age four times this amount, or 6.91 years, during the complete round trip. Now you can determine the half-way speed, using $c$ as an initial guess:

$\text{V} = \frac{1}{2} \times \text{V}$

When $\text{EQ}$ contains a list, the solver menu is built from the first object in the list. Pressing $\text{NEXEQ}$ changes the stored list by rotating its first object to the end of the list, so that the second object moves to the head of the list and is then used to build the new menu. If you look now at the SPACE variable, you will see that the equation for $V$ is the first object in the list. To restore the list to its original order, press $\text{NEXEQ}$ again.

The equation catalog operation $\text{EQ}+= $ provides a convenient means for making a list of objects for HP Solve. See section 14.2.3.2.

### 14.11 Modifying the Solver Menu

One of the advantages of HP Solve is that it automatically analyzes the current equation and builds the solver menu for you. This can, however, sometimes lead to a less-than-ideal menu. Accordingly, the HP48 gives you the option of defining the menu in a manner similar to ordinary custom menu construction (section 7.3).

Any object in EQ, or any entry in a list of objects in EQ, can be replaced by a list containing the original object plus a custom menu list:

```
{object {custom-menu}}
```

The object is used as the source for the root-finder as usual, but the normal solver menu is replaced by a custom menu defined by the inner list. The menu can have any of the features of a custom menu, with two modifications:
Menu key labels corresponding to name objects in the custom-menu list appear as black-on-white labels, and the corresponding key definitions are the usual solver menu actions.

\( E X P R = \) appears at the end of the menu.

In section 14.10 we showed how to use HP Solve with a list of related equations. In the example there, we computed the proper time \( \tau \) and the speed \( \nu \) from separate equations included in the current equation list. Since those computations required only evaluation rather than root-finding, another approach is to add menu keys for evaluating \( \tau \) and \( \nu \) to the original menu. For example, consider the following list:

\[
\{ 'X=c^2/A*(\sqrt{1+(A*T/c)^2})-1)' \]

\[
\{ 'c/A*ASINH(UBASE(A*T/c))' \quad -\text{NUM}
\quad '1_{yr}' \quad \text{CONVERT}
\}
\]

\[
\{ 'c*TANH(UBASE(A*r/c))' \quad -\text{NUM}
\}
\]

With this list as the current equation, the solver menu looks like this:

Here you can observe several things:
The custom menu list lets you place the menu keys in the order in which they are likely to be used, rather than the order defined by their appearance in the original equation.

No key is included for \( c \), the speed of light. This quantity is a constant, so there is little point in including the corresponding variable name in the menu. (Of course, you must store a value for \( c \) initially, using \texttt{STO}.)

The first three entries in the menu list are the names \( X \), \( A \), and \( T \); these define ordinary solver menu key actions.

The next two menu list entries are lists themselves. In each case, the first element is a string that is used for the menu key label. If the element were a name, the label would appear as dark letters on a white background. Using a string instead yields a label with white letters on dark, indicating that the menu key is not an ordinary solver menu key.

The second element in each individual key list specifies the unshifted menu key action. For \( \tau \) and \( V \), the actions are programs that evaluate to the values of these variables derived from previously stored or calculated variables. You can also specify left- and right-shifted menu key actions by adding additional objects to the key list (section 7.3.3).

\( \texttt{EXPR=} \) appears at the end of the menu. Pressing this key evaluates the main equation, as described in section 14.4.

If necessary, re-enter the values from the original problem:

\[
\begin{align*}
2 \text{yr} & \quad \texttt{X} & \quad 5 \text{yr} & \quad \texttt{T} \\
& & & \\
\end{align*}
\]

then

\[
\begin{align*}
\texttt{A} & : .184521099199 \_\text{ga} \\
\texttt{V} & : 4.44831376704 \_\text{yr} \\
\texttt{T} & : .318993857812 \_\text{c}.
\end{align*}
\]

As a final note to this example, we should mention that while the \textit{kinematics} of the problem--speed, time, and distance--look attractive, the \textit{dynamics}--the fuel required, radiation shielding, etc.--make such a trip effectively impossible.
15. Plotting

Much of the current interest in the use of calculators and computers in mathematics education centers around plotting. A graph of a function or of data points provides a view that is often more revealing than a symbolic expression or a table of data. Accordingly, the HP 48 provides an extensive set of automated plotting facilities that make it easy to create and work with plots. In Chapter 10 of Part I, we studied general purpose HP 48 graphics and display control, with an emphasis on the creation of special displays for program input and output. This chapter concentrates on mathematically oriented plotting, specifically the capabilities associated with the DRAW command and the plot environment.

A note on terminology: following common usage, we will use $x$ and $y$ as generic names for the variables associated with the horizontal and vertical directions on the graph screen. The HP 48 similarly uses $X$ and $Y$ as default names. However, you are by no means restricted to those choices— you can substitute any other names.

15.1 The Plot Menus

The commands and operations associated with automatic plotting are found in two menus, activated by the $\leftarrow$ PLOT and $\rightarrow$ PLOT keys. $\leftarrow$ PLOT activates the plot equation entry menu, which is a near copy of the $\leftarrow$ SOLVE menu. The $\text{NEW}$, $\text{EDEQ}$, $\text{STEQ}$, and $\text{CAT}$ operations are the same for plotting and solving, and both systems share the same current equation identified by the variable EQ. The uses of these operations are described in section 14.1.

After pressing $\leftarrow$ PLOT, you will see a display like this:

```
Plot type: POLAR
EQ: ' -2*\text{COS}(4*\text{X}) + \text{SIN}(\text{...}
4:
3:
2:
1:
```

The current equation display in the second line is the same as that described in section 14.2. The top line shows the current plot type. If the type is one of the three discrete plot types, the current equation display is replaced by a current matrix display:
The matrix is represented by its dimensions in the form \([ m \times n ]\), where \( m \) is the number of rows and \( n \) the number of columns.

The plot menu replaces the solve menu's \( \text{ROOT} \) key with \( \text{PTYPE} \). This key activates a menu that contains eight commands for selecting the eight automatic plot types. Pressing any of the PTYPE menu keys changes the current plot type to that indicated on the key label, and changes the menu back to the plot menu. The plot types are:

- **Function**: A curve \( y(x) \) is plotted by computing the vertical coordinates \( y \) for each value of the horizontal coordinate \( x \), as \( x \) is varied from the left side of the screen to the right.

- **Conic**: Both branches of the solution for \( y \) of a second-order equation in \( x \) and \( y \) are plotted in the manner of a function plot, producing a conic section figure.

- **Polar**: Points \( (r,\theta) \) specified in polar coordinates are plotted, as the polar angle \( \theta \) is varied over a specified range.

- **Parametric**: A curve \( (x(t),y(t)) \) is plotted as a parameter \( t \) is varied over a specified range.

- **Truth**: A pixel is turned on at every point \( (x,y) \) for which the evaluation of a function of \( x \) and \( y \) returns true.

- **Bar**: A bar chart is plotted that represents the values in one column of the current statistics matrix.

- **Histogram**: A histogram is generated from a column of data values sorted into equal-width bins.

- **Scatter**: Scattered points are plotted corresponding to coordinates specified by two columns of the current statistics matrix.

The first five plot types are derived from the continuous variation of a particular
variable; we will refer to these collectively as the continuous plot types. We will call the last three discrete plot types, because they use discrete individual data points.

The first key in the equation entry menu is PLOT (for “plotter”). This key activates the plot execution menu, which is also generally available with PLOT. This menu contains the actual plotting commands, ERASE, DRAW, and AUTO, plus all of the commands related to the various plot parameters—plot scale, axes, labels, etc. The relationship between the PLOT and PLOT menus is similar to that of the SOLVE and SOLVE menus. You use the PLOT menu to set up the current equation (and the plot type, for plotting), then activate the PLOT menu to do the actual plotting.

The plot execution menu has an automatic prompting display that shows several of the current plotting parameters, e.g.

![Plot type: POLAR
EQ: '-2*COS(4*θ)+SIN(...
Indep: 'θ'

\[
\begin{align*}
x: & \quad -6.5 \quad 6.5 \\
y: & \quad -1.55 \quad 1.6
\end{align*}
\]

- The top display row shows the current plot type.
- The second row shows the name and partial contents of the current equation object (for continuous plots), or the name and dimensions of the current statistics matrix (for discrete plots).
- For continuous plot types, the next row shows the name of the independent variable. For conic and truth plots, the dependent variable name is also listed. For discrete plot types, the x- and y-columns (section 15.9) and the current pseudo-linear fit model (section 20.3.2.1) are displayed on a single line.
- The current horizontal and vertical plot ranges are shown, in the format

\[
\begin{align*}
x: & \quad x_{\text{min}} \quad x_{\text{max}} \\
y: & \quad y_{\text{min}} \quad y_{\text{max}}
\end{align*}
\]

The menu commands and other operations are described in the rest of this chapter.
15.2 Plotting Essentials

When you set out to make a plot of any type on the HP 48, you must specify a number of elements that determine the nature and appearance of the graph. Obviously, you must provide either a set of coordinates for points to be plotted, or a mathematical expression from which the coordinates may be computed. But, in addition, you must determine:

1. The type of plot--the manner in which the coordinates or the expression is to be interpreted. For example, an expression might represent y as a function of x, or it might specify a radial coordinate r as a function of a polar angle θ.

2. Which variables in the expression correspond to the two coordinate axes, and the labels, if any, that are to be applied to the axes.

3. The range of an independent variable, or the portion of a discrete data set, that is to be used in a plot.

4. The plot scale, i.e. the mapping of the logical coordinates associated with a graph onto the physical display.

5. Whether axes are to be drawn, and if so, where.

6. The plot resolution, i.e. the frequency with which an expression is sampled to determine plotted points. Also, whether the calculator should “connect the dots” by drawing straight line segments between computed points.

Making a graph of \( \sin x \) is a simple example that illustrates the elementary HP 48 plotting process. Step 1 is to enter the expression, and store it in the variable EQ:

\[
'SIN(X)' \rightarrow [PLOT] \rightarrow [STEP]
\]

As shown in the example in section 14.6, the plotting command DRAW uses EQ in the same manner as HP Solve, to select an object for plotting. Since you are plotting a trigonometric function, the result will be sensitive to the current angle mode, which should be radians in this case (press \( \rightarrow [RAD] \) if necessary). Next, make sure that the HP 48 will produce an ordinary function graph (step 2):

\[
\rightarrow [PLOT] \rightarrow [FUNC]
\]

At this point, you should see a display like this:
Now enter the following:

```
  PLOT  NXT  RESET  ← PREV
```

RESET takes care of steps 3-6 by supplying default values for all of the remaining parameters.

Now you are ready to draw the graph. Press ERASE DRAW :

After drawing the graph, the HP 48 automatically activates the plot environment (see section 15.3), where you can rescale the graph (zoom), read data from the graph, and for function plots, compute intercepts, slopes, and areas. To return to the standard
15.2 Plotting

environment, press [ATTN].

In the next sections, we will study in more detail each of the graphing steps illustrated in the preceding example. We will first use ordinary function graphs to illustrate general methods, then consider other types of graphs.

15.2.1 DRAW

The center of the automatic plotting system is the DRAW command, which translates an expression or statistical data into a graph. Like HP Solve, DRAW uses an implicit argument, which may be either the object stored in the variable EQ, or the matrix stored in ΣDAT, according to the current plot type. However, DRAW also requires a number of other implicit arguments:

- The variable PPAR contains a list of parameters that determine the plot type, scale, labels, axes, and variables.
- Flag -30 determines whether DRAW plots one curve (clear) or two (set) for function and polar plots with a current equation of the form \( y = f(x) \). See section 15.4.
- Flag -31 determines whether DRAW connects successive computed points with straight line segments (flag clear), or plots unconnected points (set). The [CNCT] key in the second page of the [MODES] menu toggles this flag: when the white box appears in the menu label, the flag is clear and DRAW connects points.

(The program DRAWPIX in section 10.3.5 emulates DRAW for function plots; studying the program can help you understand the detailed behavior of DRAW.)

The [DRAW] key ([PLOT] menu) is designed for interactive plotting, and so executes three commands in sequence: DRAX to draw axes, DRAW to plot the specified data, and finally GRAPH to activate the plot environment. In a program, you can use the three commands separately or together. The [DRAW] key also provides access to the current equation: [DRAW] executes STEQ, and [DRAW] executes RCEQ.

Note that DRAW does not erase the graph screen, which makes it easy to superimpose various graphs. ERASE is a separate command; as its name suggests, it blanks the graph screen, making a clean slate for subsequent plotting. You can interrupt DRAW with [ATTN]; this terminates plotting but does not abort other program execution (e.g. DRAW still turns on the plot environment). This is helpful when DRAW is still executing, but has already plotted the useful part of a graph.
15.2.1.1 Autoscaling
When the default plot range parameters are inappropriate for a particular plot, and yet you do not know in advance what ranges might be better, you can use AUTO as a substitute for DRAW. AUTO attempts to find a vertical range such that at least a portion of the plot is visible; for some plot types, AUTO also adjusts the horizontal plot range. The algorithm is different for each plot type:

- **Function Plots.** AUTO samples the current equation at 40 points over the current x-range, discards points that return $\pm \infty$, then sets the y-range to display the maximum and minimum of the sample y values, plus the vertical origin.
- **Parametric Plots.** AUTO samples the current equation at 40 points over the independent variable range, then sets the horizontal and vertical ranges to include the minima and maxima in both directions, plus the origin.
- **Polar Plots.** Same as parametric.
- **Conic Section Plots.** AUTO only sets the vertical scale to be the same as the horizontal scale.
- **Truth Plots.** AUTO does not change the plot ranges, executing the same as DRAW.
- **Scatter Plots.** AUTO sets the x-range to the minimum and maximum values in the independent variable column (set by XCOL) in $\Sigma$DAT, and the y-range to the minimum and maximum values in the dependent variable column (set by YCOL).
- **Bar Plots.** AUTO sets the x-range from zero to the number of elements in $\Sigma$DAT, and the y-range to the minimum and maximum of the elements in the independent column.
- **Histograms.** AUTO sets the x-range to the minimum and maximum of $\Sigma$DAT elements, and the y-range from zero to the number of rows in $\Sigma$DAT.

In all cases except conic section plots and truth plots, the y-range is stretched in the negative direction by an additional 15% over that described above, to provide room for the plot environment menu labels. More details of autoscaling are provided in the sections below discussing the individual plot types.

15.2.2 EQ and $\Sigma$DAT
The five continuous plot types *function, conic, polar, parametric,* and *truth,* all use the contents of a variable EQ in the current directory to specify the plotting "source," in much the same manner as HP Solve. Following the terminology of Chapter 14, we call the source object the *current equation,* even when it is not literally an equation (which is even less likely in general for plotting than for solving). The precise interpretation of the current equation varies according to the plot type, and is described in the
corresponding sections below. The rules for indirection are the same as for HP Solve--if EQ contains a name (or a list of names), then the actual source object(s) is that stored in the named global variable(s).

When EQ contains a list of objects, each is plotted, one after another. Each object must be suitable for the current plot type. If any of the objects is a name, then the object stored in the global variable that matches the name is plotted rather than the name.

15.2.3 PPAR
With the exception of the modes established by flags -30 and -31, all of the information required by DRAW to plot the current equation is recorded in a parameter list stored in a variable PPAR (Plot Parameters) in the current directory. If PPAR does not exist when DRAW (or any other command that affects the graph screen) is executed, it is created with the following default values:

\[
\{( -6.5,-3.1) \ (6.5,3.1) \ X \ 0 \ (0,0) \ \text{FUNCTION} \ Y \}.
\]

The commands XRNG, YRNG, INDEP, DEPN, RES, CENT, SCALE, AXES, *H, and *W change the objects in the PPAR list. If the list does not exist when one of these commands is executed, the default list is stored, then modified accordingly by the command.

The list above is a particular example of the general form of PPAR:

\[
\{ (x_{min},y_{min}) \ (x_{max},y_{max}) \ \text{indep} \ \text{res} \ (x_{axes},y_{axes}) \ \text{ptype} \ \text{depn} \}.
\]

The entries are as follows:

\((x_{min},y_{min})\) is a complex number representing the logical coordinates (section 15.2.4) of the lower-leftmost pixel on the graph screen.

\((x_{max},y_{max})\) specifies the logical coordinates of the upper-rightmost pixel;

\textit{indep} is the name of the independent variable. \textit{indep} may also be a list \{ \textit{name} \ t_{min} \ t_{max} \}, where \textit{name} is the name of the independent variable, and \(t_{min}\) and \(t_{max}\) are the range of that variable over which the graph is to be plotted.

\textit{res} is the plot resolution--the increment between successive values of the independent variable.

\((x_{axes},y_{axes})\) specifies the coordinates of the intersection of the axes.

\textit{ptype} is the plot \textit{type}, actually one of the eight plot type selection commands.

\textit{depn} is the name of the dependent variable used by conic, parametric, and truth plots. For truth plots, \textit{depn} may also be a list \{ \textit{name} \ y_{min} \ y_{max} \}, where \textit{name} is the name of the dependent variable, and \(y_{min}\) and \(y_{max}\) are the range of that variable over which a graph is plotted.
The default values supplied for PPAR select the function plot type, and logical coordinates with uniform scales of 1 per 10 pixels in both directions, with the origin and axes at the center of the screen. X and Y are chosen as the independent and dependent variables, respectively. The value of 0 for the resolution (section 15.2.7) is a default for that parameter, which indicates that one point is to be plotted for every pixel column. You can restore PPAR to its default values (except for the plot type) by pressing [RESET] in the second page of the plot execution menu.

The variable ΣDAT is the analog of EQ for the discrete plot types scatter, bar, and histogram. This variable may contain a matrix, or the name of a matrix. DRAW uses some or all of the data in that matrix to create its plot. Because superimposing several discrete plots is not very useful, ΣDAT may only contain a single matrix or name.

### 15.2.3.1 Saving A Graph

The [GRAPH] key (section 10.1) makes it easy to switch between the plot environment and the standard environment, when you want to perform various calculator operations and then view a graph again. However, if you clear the graph screen or plot another graph, you must redraw the original graph if you want to view it again. One alternative is to save the picture with a sequence like PICT RCL name STO (section 10.3.3). However, this method only saves the picture itself, without recording any of the contextual information such as the plot parameters or the current equation. The programs listed below provide a mechanism for the complete recording and restoration of a graph, including the picture, the current equation, all of the plot parameters, the plotting flags –30 and –31, and the trigonometric angle mode flags –17 and –18. The latter are included because trigonometric functions are common in plotting expressions.

<table>
<thead>
<tr>
<th>PREST</th>
<th>Plot Restore</th>
<th>3DC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>level 1</td>
<td>level 1</td>
</tr>
<tr>
<td></td>
<td>'name'</td>
<td>☞</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{PREST} & \quad \text{Plot Restore} \\
\text{level 1} & \quad \text{level 1} \\
\text{'name'} & \quad \text{☞} \\
<< & \quad \text{Recall the stored list.} \\
RCL & \quad \text{Extract all of the objects.} \\
OBJ- \quad \text{Restore the current matrix or equation.} \\
DROP & \quad \text{Set full word size.} \\
\text{IF THEN} & \quad \text{Restore plot flags.} \\
\text{STO } & \quad \text{Restore plot parameters.} \\
\text{ELSE} & \quad \text{Restore plot picture.} \\
\text{END} & \quad \text{ ☞ } \\
\text{RCLF} & \quad \text{ ☞ } \\
64 & \quad \text{ ☞ } \\
\text{STWS} & \quad \text{ ☞ } \\
1 & \quad \text{ ☞ } \\
\text{GET} & \quad \text{ ☞ } \\
\#FFFFF9FFCFFFF & \quad \text{ ☞ } \\
\text{AND} & \quad \text{ ☞ } \\
\text{+} & \quad \text{ ☞ } \\
\text{STOF} & \quad \text{ ☞ } \\
\text{'PPAR'} & \quad \text{ ☞ } \\
\text{STO} & \quad \text{ ☞ } \\
\text{PICT} & \quad \text{ ☞ } \\
\text{STO} & \quad \text{ ☞ } \\
\gg & \quad \text{ ☞ }
\end{align*}
\]
PSAVE takes a name as its argument, and saves the current picture and its associated parameters in a variable with that name. PREST also takes a name, and restores the plot state that is stored with that name.

15.2.4 The Plot Scale

When dealing with mathematical plots, it is most convenient to describe positions on the graph screen in terms of logical coordinates rather than pixel numbers. As described in section 10.3.4, logical coordinates are derived from a coordinate system imposed on the graph screen, to reflect the ranges of values associated with the variables that correspond to the horizontal and vertical directions. The translation between pixel numbers and logical coordinates is determined by the first two entries in the PPAR list, \((x_{\text{min}}, y_{\text{min}})\) and \((x_{\text{max}}, y_{\text{max}})\). These specify the logical coordinates of the lower-leftmost pixel and the upper-rightmost pixel, respectively.

There are several ways to set or change the coordinate ranges. You can, of course, edit or otherwise modify PPAR directly, but this is usually less convenient than using specialized commands.

- PMIN takes a complex number and stores it as the new first element in PPAR, i.e. as \((x_{\text{min}}, y_{\text{min}})\). PMAX similarly stores a new \((x_{\text{max}}, y_{\text{max}})\) replacing the second element in PPAR. With these two commands, you can completely specify the logical coordinate system. However, the commands listed next are usually easier to use, and PMIN and
PMAX are provided primarily for compatibility with the HP 28S (they are not available in any menu).

- When you are preparing to make a graph, the most natural way to establish the coordinate system is to enter \( x_{\text{min}} \) and \( x_{\text{max}} \) together for the horizontal range, and then \( y_{\text{min}} \) and \( y_{\text{max}} \) together for the vertical range. XRNG and YRNG perform these tasks; each takes two real numbers from the stack, in the order minimum maximum, and stores them as the new real and imaginary parts of the first two PPAR elements. If you enter the parameters manually with the XRNGZ and YRNGZ keys, you get immediate confirmation since the plot menu display shows the new ranges that you just entered. You can also recall the current ranges using those keys:

\[
\begin{align*}
\text{XRNG} & \quad \text{YRNG} \\
\{x_{\text{min}} \ x_{\text{max}}\} & \quad \{y_{\text{min}} \ y_{\text{max}}\}
\end{align*}
\]

- You can specify the coordinates of the center pixel of the graph screen with CENTR, and the plotting scale using SCALE. Using these two commands fixes the plot ranges as effectively as using XRNG and YRNG. CENTR takes a complex number representing logical coordinates from the stack, and readjusts the plot parameters so that the “center” pixel will have those coordinates. The center pixel on the default graph screen has the pixel coordinates \{#65d #32d\}.

SCALE takes two real numbers as arguments. The first (level 2) specifies the new plotting scale along the \(x\)-direction, in logical units per 10 pixels (i.e. per tick mark). The second argument similarly specifies the \(y\)-scale.

Executing

\[
(0,0) \quad \text{CENTR} \quad 1 \quad 1 \quad \text{SCALE}
\]

establishes the default plot parameters, the same as executing "RESET".

The \(\rightarrow\) CENTR and \(\rightarrow\) SCALE keys return the current stored values of the center and the scale.

- \(\ast H\) and \(\ast W\) are programmable “zoom” commands that magnify or demagnify the plot scales in the \(y\)- and \(x\)-directions, respectively. See section 15.3.2.1.

- The plot environment menu (section 15.3) provides a number of interactive operations for changing the plotting ranges, using the cursor to provide the necessary coordinates.
15.2.4.1 Redimensioning the Graph Screen

When you want to extend the range of a plot without increasing the scale, you can resize the graph screen to larger dimensions than the default 131×64. In the plot environment, you can use the cursor arrow keys to scroll the display so that you can see hidden portions of the larger graph screen.

All plotting and drawing commands work normally with large graph screens, with their effects scaled to the larger size. DRAW, for example, takes longer to execute a function plot, simply because there are more columns for which to compute points than in the default case. The first and second elements in PPAR always specify the logical coordinates of the lower-left and upper-right pixels, so the conversion between logical coordinates and pixel numbers depends on the size of the graph screen.

The graph screen size can be changed by PDIM (Plot DIMensions). This command takes two arguments that specify the new horizontal and vertical graph screen dimensions. If the arguments are two binary integers \( \#dim_x \) (level 2) and \( \#dim_y \), the resulting screen is dimensioned to \( dim_x \times dim_y \) pixels, leaving the PPAR range parameters unchanged. The maximum horizontal width is 2048, but there is no limit other than available memory on the vertical height. You can also change the screen size while keeping the plotting scale unchanged, by using PDIM with complex number arguments \((x_{min},y_{min})\) and \((x_{max},y_{max})\). These arguments are stored in PPAR as the new range parameters, and the graph screen is expanded or shrunk keep the plot scales (logical units/pixel) constant: given \( w \) and \( h \) as the original graph screen height and width in pixels, the new screen will be \( w' \times h' \), where

\[
\begin{align*}
    w' &= \frac{x_{\text{max}}' - x_{\text{min}}'}{x_{\text{max}} - x_{\text{min}}} (w - 1) + 1 \\
    h' &= \frac{y_{\text{max}}' - y_{\text{min}}'}{y_{\text{max}} - y_{\text{min}}} (h - 1) + 1
\end{align*}
\]

In the plot environment, there are two ways to move the display window around on the larger graph screen:

- Move the plot cursor to an edge of the screen, then continue moving the cursor in the same direction. The cursor "drags" the window along with it. If you press \( \text{[\text{Arrow Key}]} \) before any cursor arrow key, the cursor moves to the edge of the window in the arrow direction. If the cursor is already at the window edge, it will jump to the edge of the graph screen, moving the window accordingly.

- Press \( \text{[\text{Graph}]} \). This turns off the cursor and the menu labels. The cursor arrow keys then scroll the window in the arrow directions. Pressing \( \text{[\text{Graph}]} \) again restores the cursor and the labels.
15.2.5 The Independent Variable

The third element in PPAR specifies the independent variable. This variable is the one that is varied automatically in continuous plots to produce the successive points that constitute the graphs. (For discrete plots, the independent variable is only used to label the horizontal axis.) Specifically:

- For function, conic section, and truth plots, the independent variable is associated with the horizontal axis; plotting proceeds from right to left as the independent variable is incremented.
- For polar plots, the independent variable is the polar angle.
- For parametric plots, the independent variable is not directly associated with either axis. The horizontal and vertical coordinates of plotted points are computed as separate functions of the independent variable.

The HP 48 uses the name X as the default independent variable name. INDEP allows you to change the name to any other; for example, 'θ' INDEP selects θ as the independent variable, as might be appropriate for a polar plot. A common error when you work with several different current equations and plot types is to forget that changing either does not automatically change the independent variable; you should always check its name before using DRAW. The [PLOT] menu helps here by displaying the independent variable name as part of its prompting display. Also, ZINDEPX returns the current independent variable object from PPAR.

When the independent variable is associated with the horizontal axis, its limits are obvious: its value is varied from that at the left edge of the screen to that at the right edge. DRAW uses these values (obtained from $x_{\text{min}}$ and $x_{\text{max}}$ in PPAR) as the default independent variable range for function, conic section, and truth plots. For polar plots, the default is one "circle," i.e. 0° to 360° in degrees mode, 0 to 2π in radians mode, and 0 to 400 in grads mode. For parametric plots, DRAW also uses the x-range by default, but this is seldom a useful choice.

To specify an alternate range, the independent variable entry in PPAR may be a list of the form \{ name start end \}, where name is the independent variable name and start through end is the range of values of the independent variable over which the graph is plotted. However, you don't have to re-enter the entire list to change part of it; INDEP makes various changes according to the type of its argument:

- 'name' INDEP replaces the entire independent variable entry with name alone (thus resetting to the default range).
- \{ name \} INDEP replaces the name stored in PPAR, but leaves any existing ranges unchanged. For example, if the existing entry is \{ X 0 10 \}, \{ T \} INDEP changes the
entry to \{ T 0 10 \}.

- \textit{start end} INDEP or \{ \textit{start end} \} INDEP, where \textit{start} and \textit{end} are real numbers, replace the existing range (or add a range if one is not present), without changing the name of the independent variable. Thus if the existing entry is \{ T 0 10 \} or just \textit{T}, then 5 50 INDEP or \{ 5 50 \} INDEP changes the entry to \{ T 5 50 \}. Note: LSTARG returns a list even if two separate real numbers were entered.

- \{ \textit{name start end} \} INDEP replaces the independent variable entry with the argument list.

Even for function, conic section, and truth plots, it is sometimes useful to specify an independent variable plotting range that is different from the \textit{x}-range defined by the screen limits. For example, if you are plotting \( \sqrt{X} \) with the default \textit{x}-range from \(-6.5\) to \(+6.5\), you might set the independent variable range from 0 to 6.5, to avoid unnecessary time spent “plotting” at negative values of \( X \) where the expression returns imaginary values.

### 15.2.6 The Dependent Variable

The seventh entry in the PPAR list specifies the \textit{dependent variable}, which is used as the second variable required for conic section, parametric, and truth plots. For other plot types, the dependent variable name is used only to label the vertical axis. The dependent variable object is either a name, or a list \{ \textit{name start end} \}. \textit{Start} and \textit{end} are used by truth plots to specify the range of \textit{y}-values for the plot.

DEPND establishes a new dependent variable entry from an argument that may be a name, a list, a complex number, or two real numbers, following the same logic as INDEP. \texttt{DEPNZ} returns the current dependent variable object to the stack.

### 15.2.7 Resolution

The fourth entry in PPAR is a real number or a binary integer that specifies the plot \textit{resolution}. In general terms, this parameter determines the increment in the independent variable between successive points that are plotted. If it is a real number, the increment is in logical units; a binary integer specifies an increment in pixels. The precise meaning varies according to the current plot type; we will postpone detailed explanations to the sections describing the individual plot types.

The initial default value of the resolution parameter is 0. This value is often the most convenient, since it gives reasonable results for all of the plot types, so you don’t have to change it when you switch types. When necessary, you can change the resolution with the command RES, which takes a real number or a binary integer and stores it as the new fourth entry in PPAR. \texttt{RES} is found in the second page of the plot execution
Plotting 15.2

returns the current resolution parameter to the stack.

Table 15.1 below summarizes the interpretation of the resolution, and the meaning of the default value 0, for each of the eight plot types.

Table 15.1. Plot Resolution

<table>
<thead>
<tr>
<th>Plot type</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>real step = 0</td>
</tr>
<tr>
<td>Conic</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Polar</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Parametric</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Truth</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Scatter</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Histogram (bin width)</td>
<td>real step ≠ 0</td>
</tr>
<tr>
<td>Bar (bar width)</td>
<td>real step ≠ 0</td>
</tr>
</tbody>
</table>

In the table, x is the horizontal coordinate (and the independent variable for function, truth, and conic plots); t is the independent variable for parametric plots; θ is the independent (polar angle) variable for polar plots. scalex is the plot scale in the x-direction, logical units per 10 pixels. An increment Δx = scalex/10 is equivalent to moving one pixel width in the x-direction.

Leaving the resolution at its default value of zero is often a good choice, since then you don't have to worry about changing the resolution when you switch plot types or current equations.

15.2.8 Axes and Labels

When you execute DRAW by means of the DRAW key, the HP 48 automatically draws axes on the graph screen to help you interpret the graph. DRAW actually invokes DRAX (DRaw AXes); to reproduce the key's effect in programs, you must include DRAX as well as DRAW. The axes are drawn through the point specified by the fifth element in PPAR. Usually, this element is a complex number that represents the coordinates of the origin of the axes. It may also be a list of the form

\[ \{ (x_{axes}, y_{axes}), "label_x", "label_y" \} \]
15.2 Plotting

Here the first element specifies the intersection of the axes, and the remaining two elements are strings that are used to label the x- and y-axes, respectively.

AXES allows you to move the axis origin to another position, or to add or change axis label strings to PPAR. This command takes a variety of arguments, much like INDEP or DEPND:

- If the argument is a single complex number, it replaces the current PPAR entry.
- If the argument is a list containing only a complex number, that number replaces the current origin coordinates without affecting any current label strings.
- If the argument is a list containing two strings, the strings replace the current label strings (the PPAR element becomes a list, if it is not already). The origin is unchanged.
- If the argument is a list containing a complex number and two strings, the list replaces the current PPAR entry.

The actual labeling is performed by LABEL, which you may execute as an ordinary command or by means of the \( \text{LABEL} \) keys in the third page of the plot execution menu and in the first page of the plot environment menu. LABEL labels the axes with variable names or strings, and with the values of the coordinates at the edges of the plot. Labels are placed on the graph screen like this:

![Graph Screen with Labels](image)

This picture was generated by the following sequence:

```
STD 0 10 XRNG -5 5 YRNG (5,0) AXES
{ "XNAME" "YNAME" } AXES DRAX LABEL
```

STD is included in the sequence because LABEL formats its coordinate values according to the current real number display mode.
LABEL will make axis labels even if you haven’t stored label strings in PPAR. The horizontal axis label shown as XNAME in the picture is chosen in the following priority order:

1. The "label_x" string.
2. The independent variable name.

(The latter choice is usually not too meaningful for polar or parametric plots.) The vertical axis name is similarly chosen in the following priority order:

1. The "label_y" string.
2. If flag -30 is clear, and the current equation is of the form name = expression, where name is not the independent variable, then name.
3. The dependent variable name from PPAR.

15.2.8.1 Tick Marks

The tick marks that are drawn on the axes by DRAX are placed at ten-pixel intervals (measured from the axis origin). With the default plot scales, the marks are spaced one logical unit apart; with other scales they may or may not be helpful as position indicators other than to distinguish the axes from other lines.

Despite the modest pixel density of the HP 48 display, tick marks at other spacings can be useful, even if they end up with irregular spacing. The programs listed next let you specify arbitrary and independent tick spacings for the horizontal and vertical axes (SETTICKS), and to draw axes with the custom tick marks (TICKAXES). The tick spacings, which you may specify with real numbers (logical units) or binary integers (pixels), are stored as a two-element list in a variable TICPAR.

<table>
<thead>
<tr>
<th>SETTICKS</th>
<th>Store Tick Spacings</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 2</td>
<td>level 1</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>#n_x</td>
<td>#n_y</td>
</tr>
</tbody>
</table>

\[ \text{\texttt{\< 2 \rightarrow LIST}} \]
\[ \text{\texttt{'TICPAR' \rightarrow STO}} \]

Combine parameters into a list.
Store as TICPAR.
15.2 Plotting

**Tickaxes**

```
<< (0,0) PIX? DROP
PPAR OBJ- DROP
TICPAR
IF DUP TYPE 6 SAME
THEN 1 DUP SETTICKS RCL
END
OBJ- DROP
IF DUP TYPE 10 ==
THEN #0 2 -LIST PX-C {#0 #0} PX-C - IM
END SWAP
IF DUP TYPE 10 ==
THEN #0 2 -LIST PX-C {#0 #0} PX-C - RE
END
9 ROLL C-R 10 ROLL C-R
9 ROLL
IF DUP TYPE 5 == THEN 1 GET END
C-R
- dy dx xmin ymin xmax ymax xaxis yaxis
<< 4 DROPN
xmin yaxis R-C xmax yaxis R-C LINE
xaxis ymin R-C xaxis ymax R-C LINE
xaxis yaxis R-C DUP C-PX
OBJ- DROP 1 - SWAP 1 + SWAP 2 -LIST
PX-C
DUP2 + 3 ROLLD -
C-R ROT C-R -> xm ym xp yp
<< xmin xaxis - dx / 0 RND dx * xaxis + xmax
FOR x x yp R-C x ym R-C LINE
dx
STEP
ymin yaxis - dy / 0 RND dy * yaxis + ymax
FOR y xp y R-C xm y R-C LINE
dy
STEP
>>
>>
```

Ensure PPAR exists.
Take PPAR apart.

If TICPAR doesn’t exist, then create a default version.
Take TICPAR apart.
If y-increment is binary, convert to real.
If x-spacing is binary, convert to real.

Axes entry.
Extract coordinates from list.

Store in local variables.
Discard extra parameters.

Draw x-axis.
draw y-axis.
Compute x- and y-coordinates of points next to the axes.

\[
\begin{align*}
(x_{\text{axis}}, y_{\text{axis}}) & \quad (x_{\text{axis}} + y_{\text{axis}} +) \\
(x_{\text{axis}} + y_{\text{axis}} +) & \quad (x_{\text{axis}} - y_{\text{axis}} -)
\end{align*}
\]

Save coordinates in local variables.
Position of first x tick.

Draw the next x tick.
Position of first y tick.

Draw the next y tick.

15.2.9 Plot Type
The sixth PPAR entry is one of the eight plot type commands FUNCTION, CONIC, POLAR, PARAMETRIC, TRUTH, SCATTER, BAR, and HISTOGRAM. When any of these commands is executed, it stores itself as the sixth entry in the PPAR list. There, the command name is used by DRAW to determine the type of plot to create.
15.3 The Plot Environment

When you press [GRAPH] (or just [\] when no command line is present) or [DRAW], the HP 48 switches to the *plot environment* (see section 6.1), where the keyboard and display are dedicated to interaction with the graph screen. Here you can view graphs, add labels, points, lines, and other geometric figures, change the plotting ranges and scales, transfer coordinates and pictures to and from the stack, and perform various analytic operations on function graphs. As from other HP 48 environments, you can exit back to the standard environment by pressing [ATTN].

To illustrate the use of the plot environment, create a simple function plot:

```
STD RAD 'SIN(10*COS(X))' \[PLOT\] NEW ENVEX ENTER
```

The display is characteristic of the plot environment:

- The upper part of the display shows the graph screen, currently with the graph of $\sin(10\cos(x))$.
- The first page of the plot environment menu appears in the menu label area.

The menu labels actually overlay part of the graph screen. For example, if you press [LABEL], you may notice that you can only see three of the four coordinate-limit values:
To uncover the hidden part of the graph, press \[ - \] (to “take away” the labels), or \[ KEYS \] in the third page of the menu. To restore the labels, press any menu key, or \[ + \] again. You can also move the cursor downwards with \[ V \] or \[ \rightarrow V \]. When the cursor reaches the top of the menu labels, further downward motion causes the graph screen to scroll upwards, revealing the hidden part. Finally, you can press \[ ON GRAPH \], which removes the menu labels and the cursor. This is especially useful when you want to use \[ ON PRINT \] to print an image of the graph screen without the extraneous features.

Along with the menu keys (including \[ NXT \], \[ PREV \] and \[ PREV \]), the rest of the HP 48 keyboard is redefined for plot-related operations:

- \[ A \], \[ < \], \[ V \], and \[ \rightarrow \] move the plot cursor in the indicated directions. Pressing \[ \rightarrow \] before any of the arrow keys moves the cursor to the edge of the screen in that direction.
- \[ STO \] copies the graph screen to the stack as a graphics object.
- \[ ENTER \] returns the cursor position to the stack as a complex number object.
- \[ +/- \] switches the cursor type.
- \[ DEL \] erases a rectangular portion of the graph screen.
- \[ \rightarrow CLR \] erases the entire graph screen.
- \[ \times \] sets the plot mark.
- \[ - \] turns the menu labels on and off.
- \[ + \] turns the coordinate display on and off.

### 15.3.1 The Plot Cursor
Not immediately visible in the display picture above is the *plot cursor*, which is initially placed at the center of the screen. To reveal the cursor, you can move it away from the axes with \[ \Rightarrow \] and \[ A \]:

![Plot Cursor Diagram](image-url)
Now you can see that the cursor is a small "+". The cursor is used as a focal point for various operations. For example, if you press \texttt{COORD} or \texttt{+}, you can read the cursor position in the menu label area:

![Cursor Coordinate Display]

The coordinate display follows the current number display format, which we set to standard format at the start of this example. If you move the cursor while its coordinates are being displayed, the coordinates are changed along with the cursor position (this slows cursor movement down somewhat). You can restore the menu labels by pressing \texttt{+}, \texttt{-}, or any menu key.

For use in further computations, you can copy the cursor position to the stack by pressing \texttt{ENTER}. You will see the busy annunciator flash momentarily, but there is no other visible indication until you exit the environment by pressing \texttt{ATTN}:

![Cursor Coordinate as Complex Number]

Here you can see the cursor coordinates represented as a complex number. To return to the plot environment now, press \texttt{<}.

Because the default plot cursor is drawn as a superimposed cross, it is invisible any time it is situated on a dark screen region. This makes the cursor particularly hard to use with bar charts and histograms. To remedy this situation, you can alter the nature of the cursor by pressing \texttt{+/−} (or \texttt{+/−} in the third page of the plot environment menu).
This changes the cursor from a simple superposition of the cross to a pixel-by-pixel reversal, in the manner of GXOR (see section 10.3.1). On a dark background, the cursor is white, whereas on a white background, the cursor is dark. In an area with a mixture of dark and white pixels, you can press \( \pm/\mp \) several times to see where the cursor is. The cursor type is encoded by flag \(-32\), which allows you to set the type in a program. The default superimposed cursor corresponds to flag \(-32\) clear.

### 15.3.1.1 The Mark

For certain operations, it is necessary to specify two points on the screen. The cursor is always available as one of those points; for the other, you set a mark. The mark is set by moving the cursor to a point, then pressing \[\times\] or \[\text{MARK}\] (in the third page of the plot environment menu). This draws an \(\times\) mark, which remains when you move the cursor away. If a mark already exists, the old mark is erased and a new one drawn—unless the cursor is actually on the mark, which just erases the mark.

If you press any menu key that requires a mark when no mark exists, that key will set the mark rather than executing its labeled operation.

A simple example of a cursor/mark operation is \[\text{DEL}\], which erases the graph screen in the region defined by the cursor and the mark. With the cursor and mark as follows,

\[\text{DEL}\] erases like this:
This operation is also available via the menu key DEL, in the third page of the plot environment menu.

15.3.2 Recentering and Zooming

Unless you are clever in choosing the plot ranges before you make a graph, chances are the plot ranges will not be quite right. It is often easier to make a trial plot, then adjust the ranges, than it is to choose the ranges in advance. The key operations for this purpose are ZOOM and Z-BOX, found in the first page of the plot environment menu.

Recentering is a simple one-step operation by which you can move a graph relative to the screen so that some desired feature is centered on the screen. All you do is move the cursor to the point where you want the new screen center, then press CENT. For example, with the cursor as shown here

![Recentering example](image)

**CENT** redraws the picture like this:

![Recentered graph](image)

You could achieve the same effect by pressing ENTER (to enter the cursor coordinates), then ATTN CENTR ERASE DRAW, but for interactive use, performing the recentering from within the plot environment is more convenient.
The term “zooming” originated in photography, where use of variable focus lenses allows you to change the magnification of a scene. In computer graphics, it has come to mean a similar change of magnification of a plot, even though the speed implied by the term may not be possible. For a computed plot, there is seldom any advantage to changing the plot scales unless the plot is recomputed to add new data.

In the plot environment, you can zoom in on or out from a plot by specifying numerical magnification factors, or, for zooming in, by selecting a region that you want to expand to fill the screen. The latter is the most intuitive method; you position the mark and the cursor to define a rectangular region, then press `Z-BOX` (Zoom-to-BOX). The plot is redrawn so that the region fills the screen (the full 131×64). For example, with the cursor and mark as shown,

```
X
200t
£Z-BOXE yieldsthispicture:
```

You can use `Z-BOX=` to change the scale in only one direction by placing the mark and the cursor in the same row or column. That is, if the mark and the cursor are in the same column, `Z-BOX=` increases the y-scale to stretch the marked range to fit the screen vertically, but leaves the x-scale unchanged. Similarly, using `Z-BOX=` with the
cursor and the mark in the same row increases the x-scale but does not change the y-scale. (If the cursor is on the mark, Z-BOX does nothing.)

When you want to zoom out, i.e. increase the plot scale so that more of a curve or other structure fits on the screen, you must use the numerical ZOOM operation. Here you may specify a magnification factor by which to multiply the x- or y-scale, or both; a factor larger than 1 zooms out, shrinking plot features by that factor. A factor smaller than 1 zooms in, increasing the size of plot features.

With the picture as shown above, pressing ZOOM changes the menu as shown here:

EXIT allows you to cancel the change, returning to the main plot environment menu without altering the plot. Pressing any of the other four labeled menu keys prompts you for a zoom magnification factor. For instance, pressing X produces this display:

Now you may type in a zoom factor, or press ATTN to return to the zoom menu. Try 2 ENTER: 

-471-
The plot is redrawn, with a doubled x-scale. The logical coordinates of the center of the screen are preserved, so that the curve appears to be compressed along the x-direction, around the screen center. An analogous behavior is produced by pressing $\text{Y}$ in the zoom menu, with the y-scale varying according to the zoom factor and the x-scale held constant. $\text{XY}$ produces a uniform rescaling in both directions by the entered factor. The latter is often the most useful when you are searching for a particular plot feature; it lets you zoom out, repeatedly if necessary, until you can see the feature. Then you can zoom back in to magnify that area of the graph.

The final option in the zoom menu is $\text{EXAUTO}$. This operation rescales in the x-direction according to the zoom factor you enter, then autoscales in the y-direction, following the autoscale logic described in section 15.2.1.1.

15.3.2.1 Programmable Zooming
The commands $\text{H}$ and $\text{W}$ are programmable commands that can stretch or shrink a plot in height and width, respectively. For either command, you supply a real number argument that is used by the command to multiply the appropriate coordinates of $P_{\text{min}}$ and $P_{\text{max}}$. For example, to double the height of a plot (i.e. to flatten a curve to half its original height on the screen), you execute 2 $\text{H}$, then DRAW. To make a plot cover twice the horizontal range of the previous plot (compressing the plotted curve sideways), execute 2 $\text{W}$. The effects are the same as manual zooming using the ZOOM menu in the plot environment, with zoom factors of 2.

The effects on the plot range parameter of using $\text{H}$ and $\text{W}$ with arguments $h$ and $w$ are listed below. The primed quantities are the new values, and the unprimed quantities are the original values.

$\text{H}$

\[
y'_{\text{min}} = \frac{(32 + 31h)}{31} y_{\text{min}} + \frac{(31 - 31h)}{31} y_{\text{max}}
\]

\[
y'_{\text{max}} = \frac{(32 - 32h)}{31} y_{\text{min}} + \frac{(31 + 32h)}{31} y_{\text{max}}
\]
\[ x'_{\min} = \frac{1}{2}(1 + w) + \frac{1}{2}(1 - w) x_{\max} \]
\[ x'_{\max} = \frac{1}{2}(1 - w) + \frac{1}{2}(1 + w) x_{\max} \]

### 15.3.3 Drawing on the Graph Screen

In addition to the labels you can add automatically using LABEL (section 15.2.3), the plot menu provides several other tools for embellishing a graph and for making diagrams that may or may not be related to any mathematical plot. These tools are interactive versions of the drawing commands described in section 10.3.4, using the cursor and the mark to specify positions rather than stack-entered coordinates.

To illustrate these operations, we will start with a blank graph screen. To follow the examples, you should exit from the plot environment by pressing \boxed{\text{ATTN}} (if necessary), then \boxed{\text{PLOT} \text{ ERASE}} , and \boxed{\text{<}} to return to the plot environment. (You can also clear the screen without leaving the environment by pressing \boxed{\text{PLOT} \text{ CLR}} , but the preceding method also returns the cursor to the center.)

The simplest operation is turning “on” (dark) a pixel, as you might with PIXON. This is done by pressing \boxed{\text{DOT+}} (second page) twice, which turns on the pixel at the center of the cursor. If you do this then move the cursor a few pixels to the right, you will see the center pixel turned on:

![Blank Graph Screen with Pixel Turned On]

Pressing \boxed{\text{DOT+}} just once activates a line-drawing mode in which the cursor leaves a trail behind as you move it. Thus

\boxed{\text{DOT+}} \text{ A A A A A A < < < < < < < < <}

draws lines like this:
Notice that the DOT+ key label contains a white square when the line drawing mode is active. Pressing DOT+ when the square is visible turns the mode off and removes the square from the label.

DOT- turns the cursor into a pixel "eraser." Pressing this key once erases the pixel at the cursor center, and turns on a white square in the key label (turning off the square in the DOT+, if present). While in this mode, the cursor erases pixels as you move it. Pressing DOT- a second time deactivates the eraser mode.

For drawing lines in arbitrary directions, LINE is faster and more accurate than using DOT+. LINE connects the cursor and the mark with a straight line. The mark is automatically moved to the cursor position, which makes it easy to draw a series of connected lines:

For drawing lines through already-dark regions, TLINE reverses the pixels along the line connecting the cursor and the mark:
Plotting

BOX is the interactive counterpart of the programmable BOX, drawing a rectangle defined by the cursor and the mark at opposite corners. With the cursor and mark positioned like this:

BOX draws a rectangle:

The final simple drawing tool is CIRCL, which is the interactive version of ARC. This
operation nominally draws a circle, with its center at the mark. For example, with the cursor and mark still on the corners of the box just drawn, \( \text{CIRCL} \) draws this circle:

To draw only a portion of a circle, place the mark at the center of the circle, and the mark at the start of the arc you wish to draw. Then press \( \text{CIRCL} \) ; when the arc is drawn as far as you want, press [ATTN] to stop the drawing.

15.3.4 Working with Graphic Objects

The graphics operations embodied in the commands STO, SUB, and REPL, as they apply to the graph screen via PICT (section 10.3.3), are available as interactive operations in the plot environment menu. As for the pixel commands described in the previous section, the mark and cursor are used in lieu of the screen coordinates used as command arguments.

Pressing [STO] in the plot environment enters a copy of the graph screen into stack level 1 as a graphics object. The object contains the full graph screen picture, but without the menu labels, the cursor, or the mark. The action of [STO] in the plot environment is apparently reversed from its action using PICT in the standard environment. That is, plot environment [STO] is equivalent to the command sequence PICT RCL. This reversal arises from the “direction” of the store: in the standard environment, the stack object is the visible focus, and you store that object into the graph screen (PICT). In the plot environment, the graph screen is visible, and [STO] stores the picture onto the (invisible) stack.

[STO] RCL does not work in the plot environment, but you can bring a graphics object from the stack to the graph screen by using [REPL] , in the third page of the menu. This operation takes a graphics object from the stack and writes it onto the graph screen, replacing the screen region with the cursor at its upper-left corner. [SUB] provides the reverse operation; it copies a rectangular region defined by the mark and the cursor to the stack as a graphic object. These operations are easily illustrated by a simple example. Starting in the plot environment with a blank screen (use [STO] CLR ), place
the cursor and the mark as follows:

Then

 copies the region bounded by the mark and the cursor to the stack as a graphics object, where (NEG) reverses the image. Then, back in the plot environment, replaces a region below and to the right of the cursor, in this case with a dark rectangle. The region affected by is the same size as the stack object, except that any rows or columns in the graphics object that would extend beyond the edges of the graph screen are truncated.

### 15.4 Function Plots

The plot of $\sin x$ used as an example in section 15.2 is a typical example of a function plot. In this type of plot, the $y$-coordinate is a function of the $x$-coordinate, where each value of $x$ yields a single value of $y$. In particular, the $x$-coordinate is incremented from the left edge of the graph screen to the right, in steps determined by the resolution parameter (section 15.2.7); one point is plotted in each column corresponding to the
successive values of \( x \). If the resolution is a real number, \( x \) is incremented by that amount; if it is a binary integer \( \#n \), \( x \) is incremented by \( n \) pixel columns. (The default value of 0 is equivalent to a value of \( \#1 \), for plotting in every column.)

If the curve-fill flag (-31) is clear, computed points separated by more than one pixel in either direction are connected by a “straight” lines to make continuous plotted curves. Although plotting these filled curves is the default choice, you should use some care in using that option, since it causes many points to be plotted that are not computed directly from the current equation, which can be misleading.

The function \( f(x) \) that determines the \( y \)-coordinates of plotted points is specified by the current equation. In its simplest form, this object is an expression in the independent variable \( x \). However, the current equation may also be any of the following:

- **Multi-variable expressions.** HP 48 expressions can contain any number of variables, e.g. \('\text{SIN}(A+B\times X)^C'\). One variable is designated (by INDEP) as the independent variable; you must supply values for the remaining variables so that the expression evaluates to a number. Note that since HP Solve uses the same current equation in EQ, you can use the solver menu (section 14.0) to store values for all of the variables.

- **Programs.** Any program equivalent to an expression (takes no arguments from the stack, and returns one number) can be used instead of an expression.

- **Definition Equations.** When plotting functions, we commonly speak of plotting \( y=f(x) \), where \( f(x) \) is the function, and \( y \) is little more than a label for the vertical axis. To accommodate this convention, when the current equation object is an equation of the form \('\text{name}=\text{expression}'\), and flag -30 is clear, DRAW graphs as if the object were just \('\text{expression}'\). Thus, plotting \('F=\text{SIN}(X)'\) or \('G=\text{SIN}(X)'\) produces the same curve. The only difference is that LABEL labels the vertical axis with \( F \) in the first case and \( G \) in the second. Setting flag -30 cancels this special treatment of defining equations, which are then plotted as two separate expressions, as described below.

- **Vertical lines.** When the current equation has the form \('x=constant'\), where \( x \) is the independent variable, and flag -30 is set, DRAW plots a straight vertical line. With flag -30 set, DRAW plots a horizontal line at \( y=constant \), and a diagonal line \( y=x \).

- **Other Equations.** When the current equation is a true equation, that is, two expressions combined with an “=” sign, the HP 48 graphs the two sides of the equation as separate curves, regardless of the state of flag -30.

The treatment of equations as separate expressions arises from the deliberate
connection between the HP 48's plotting and solving systems. The solutions obtained by HP Solve correspond to the intersection points of the two curves. This does provide a simple method of graphing two expressions simultaneously; you can form an equation by setting the two expressions equal to each other, then graph the equation. For example,

```
RAD ▶ PLOT ▶ NXT ▶ RESET ▶ ◀ PREV
'COS(X)=1/X' ◀ DRAW ▶ ERASE ▶ DRAW
```

yields the following display:

![Graph displaying two expressions plotted simultaneously.]

Note that the two expressions are plotted simultaneously, unlike the case when EQ contains a list of expressions, for which the curves are plotted sequentially.

Although function plots are nominally designed to plot y as a function of x, DRAW does make special provision for current equations of the form \( x=constant \), where \( x \) is the independent variable. When flag -30 is clear, such equations are plotted as vertical lines with \( x \)-intercept at \( constant \). Thus 'X=2' plots as

![Graph showing a vertical line at x=2.]

If flag -30 is set, the same equation plots as
The horizontal line is the right side of the equation, representing the line $y = 2$. The diagonal line is the right side, corresponding to $y = x$.

15.4.1 Plotting Programs
As stated previously, a program used as a "current equation" should execute as if it were an algebraic object, taking no arguments and returning one real number. However, as long as the program follows this input/output convention, it can perform any number of additional operations on the stack or memory during its execution. There are two restrictions:

- The program can not halt using HALT or PROMPT. Executing either of these commands causes the HALT Not Allowed error.
- After completing its plot, DRAW restores the stack to the same number of objects that were present when it started (this is done as part of DRAW's error-handling process). If a program is to return extra results that you want to keep after plotting, it must store them in a variable or use some other strategy to protect them from the stack reset.

You should note that a user-defined function (section 8.5) is a program, and moreover, one that uses stack arguments. Therefore, you cannot plot a user-defined function by storing the program or its name in EQ. Instead, you must include it as part of an expression, with the independent variable as its argument. That is, if you have a user-defined function named $F$, you can plot '$F(X)$', but not '$F$' by itself.

The programs listed next give examples of what you can do by using a program as the current equation object. The first, MANPLOT, lets you plot points manually, one at a time. When you execute DRAW with MANPLOT in EQ, you will see the graph screen briefly, then a display like this:
At this point, you type the y value corresponding to the x value in the prompt, and press \texttt{ENTER}. \texttt{DRAW} plots the specified point, then prompts you for the next one, and so on.

The next program, \texttt{SCDRAW}, is a replacement for \texttt{DRAW} that saves the coordinates of all of the computed points as it plots them. Upon completion, \texttt{SCDRAW} returns a list containing all of the points, in order. \texttt{SCDRAW} is designed for function plots where the current equation is an expression or a program. If the current equation is an equation, the y-coordinates returned by \texttt{SCDRAW} will be the differences between the y-coordinates computed from the the left and right sides of the equation.
SLVDRAW uses an approach similar to that of SCDRAW, in this case to let you plot the solutions of an equation, rather than treating an equation as two independent expressions. For each value of the independent variable, SLVDRAW uses ROOT to find a value of the dependent variable, and plots the corresponding point. Before plotting, you must store in the dependent variable an estimate of its value at the left end of the screen; subsequently, each solved value is used as an initial guess for the next solution. As an example,

\[ X^2 + Y^3 = 9 \]  STEQ  -3  'Y' STO SLVDRAW

with the default plot ranges yields this plot (after adding labels):
\[ \begin{align*} &<< \text{RCLF } -31 \text{ CF} \\
&\text{IFERR 'PPAR' RCL} \\
&\text{THEN 1 *H RCL} \\
&\text{END} \\
&7 \text{ GET} \\
&\text{IF DUP TYPE 5 ==} \\
&\text{THEN 1 GET} \\
&\text{END} \\
&'EQ' \text{ RCL } \rightarrow \text{ flags dep eq} \\
&<< \text{IFERR} \\
&\langle= \text{ eq dep DUP EVAL ROOT} \\
&\rightarrow 'EQ' \text{ STO} \\
&\text{DRAX DRAW} \\
&\text{THEN 1} \\
&\text{ELSE 0} \\
&\text{END} \\
&\text{eq 'EQ' STO flags STOF} \\
&\text{IF THEN ERRN DOERR END} \\
&\rangle \\
&\rangle 
\end{align*} \]

Activate argument recovery.

Get PPAR list.

Get dependent variable name.

Save in local variables.

Solve for the dependent variable.

Replace EQ with program.

Make the graph.

Signal that an error occurred.

Restore EQ and flags.

Report any error.

15.4.2 The Function Menu

The \texttt{FNC} menu key in the plot environment activates a sub-menu of operations that are applicable to function-type plots (for other plot types, \texttt{FNC} flashes \texttt{Invalid PTYPE}). In general, the operations are designed for extracting information from a function curve at one or more points, where the points in question are specified by means of the cursor.
To help in exploring the operations, we will use a list of expressions (stored as LIST.EQ) as the current equation:

\[
\{\ '\text{SIN}(X)'\ '\text{COS}(X)'\ '(X/3)^2-1'\ \}\ \text{NEW} \ \text{LIST} \ \text{ENTER}
\]

'PPAR' PURGE -1.55 1.6 YRNG RAD ERASE DRAW

then produces this picture:

![Graph of the expressions]

When the current equation is a single object, all of the function menu operations are directed to that object. If you are working with a list of objects, all of the operations select the first object in the list; ISECT may also use the second object. To determine which object is the first in the list, you can either

- press \[\text{Review}\] to show the beginning of the equation list, which is usually enough to identify the first object; or

- (in the FCN menu) press \[\text{NXT} \ \text{F}(X)\]. This displays the value of the selected object at the horizontal position of the cursor, and moves the cursor to the selected curve.

In the current example, moving the cursor a few pixels to the right, then pressing \[\text{F}(X)\] moves the cursor to the \('\text{SIN}(X)'\) curve:

![Cursor on 'SIN(X)']
At the end of the second menu page is the key \texttt{NSEQ}, which selects the next object in the current equation list:

\begin{center}
\texttt{\textcolor{red}{\textbf{NSEQ}}}
\end{center}

Notice that \texttt{NSEQ} also displays the newly selected object at the bottom of the screen. Using this operation, you can cycle through all of the objects in the current equation list. It works by actually moving the first object to the end of the list, so that the second object moves to the front.

Returning to the first page of the menu, pressing \texttt{ROOT} finds the zero of \( \cos(x) \) at \( \pi/2 \):

\begin{center}
\texttt{\textcolor{red}{\textbf{ROOT}}}
\end{center}

\texttt{ROOT} invokes the root-finder on the selected object, using the horizontal cursor position as an initial guess. When it has finished, it displays the result at the bottom of the screen and move the cursor to the root.

Now press any key to restore the menu, then \texttt{ISECT}:
ISECT finds the intersection of the curves corresponding to the first two objects in the current equation list:

- When the first object in the list is an actual equation, ISECT finds the intersection of the two expressions that comprise the left and right sides of the equation; ISECT is the same as ROOTE in this case.
- When the first two objects are both expressions, ISECT finds their intersection by combining them into an equation and invoking the root-finder.
- When the first object is an expression, and the second is an equation, ISECT substitutes the first expression for the left side of the equation, and solves the resulting equation.

The next entry in the function menu is SLOPE. This operation computes the slope of the selected curve at the x-position of the cursor:

It does this by symbolically differentiating the selected object, then numerically evaluating the result. Differentiation is also used by EXTR, which finds the critical points of a curve. Pressing EXTR here finds the local minimum of the cosine curve at $\pi$: 
EXTRM differentiates the selected object, then applies the root-finder to the result to find a critical point where the derivative is zero.

AREA uses the HP 48's numerical integration facility (section 18.4.2) to find the area of a curve between two x-values selected by the cursor and the mark. For example, to find the area of the “hump” of the cosine curve between its zeros at ±π/2:

1. Move the cursor near the zero at π/2, and press \texttt{\textsc{root}} \times \times (the first \times just restores the labels).
2. Move the cursor near the zero at −π/2, and press \texttt{\textsc{root}}  .
3. Press any key, then \texttt{\textsc{area}} :

The difference between this result and the ideal result −2 (negative because we integrated from right to left) arises primarily because the integration limits are the x-coordinates of the centers of the pixels near the zeros, rather than the computed coordinates. If you execute \int using the values returned by \texttt{\textsc{root}} (which are actually on the stack), you do obtain a result of 2.

The remaining FCN menu operation is \texttt{\textsc{f}} (on the second page). This operation adds the derivative of the selected object to the head of the current equation list, then erases
the screen and executes `DRAW` again. For example, press `FXEQ` to select the parabola, then `S`:

![Parabola Plot](image)

The derivative of the parabola is a straight line with slope 0.5.

If you exit to the stack at this point (`ATTN`), you will see a display like this:

![Stack Display](image)

All of the function menu operations that compute results for display in the plot environment also return those results to the stack. Each result is tagged with the operation name to help identify it.

### 15.5 Conic Sections

HP 48 function plots normally do not attempt to plot solutions of equations (see section 15.4.1), because the solving process is relatively slow (using a root-finder to compute each point), and because there may be multiple solutions. However, in the case of general quadratic equations, it is possible to solve each equation symbolically, and to find both branches of the solution. This is the basis of the conic section plot type, so named because the graphs of second-order equations are called conic sections.

The general procedure for making a conic section plot is the same as for a function plot,
where in this case you must execute CONIC, or use CONIC in the PTYPE menu, to select the conic plot type. The current equation must be an algebraic object, which should be an expression or equation in two variables, nominally second order in both. If you wish to use names other than X and Y as the independent and dependent variables, you can use INDEP and DEPND to specify alternate choices.

You can understand what the HP 48 does to make a conic plot by considering how you might do it yourself using function plots. Start with a general second-order equation:

\[ A*X^2 + B*Y^2 + C*X + D*Y + E*X*Y + F = 0 \]

Applying 'Y' QUAD (section 16.4.3) to this equation to solve for Y yields (after COLCT)

\[ Y = \frac{0.5 \sqrt{(-4*(A*X^2 + C*X + F) + B) + (E*X + D)^2} * s1 - E*X - D}{B} \]

As you would expect for a quadratic equation, the solution is double-valued, as indicated by the presence of the s1 that represents ±1 (section 16.5). If you store 1 in s1, you get an equation for one solution branch; storing −1 in s1 yields the other. To plot both branches of the solution simultaneously, you can use a current equation formed by equating the two right-sides of the separate branch equations, then execute an ordinary function plot using DRAW. This is essentially what the HP 48 does for conic plots, except that it makes a temporary version of the two-branch equation rather than disturbing the contents of EQ.

To try out conic plotting, store the general equation above in EQ and use the solver menu to store various values for the coefficients A-F, then execute DRAW. For example, with A=1, B=2, F=-9, and zero for the remaining coefficients, you obtain an ellipse (here we are using the default plot ranges):

There is a delay before you see the ellipse drawn. This happens because values of X to the left and to the right of the ellipse give imaginary solutions for Y, for which no points are plotted. Making E=1 produces a rotated ellipse with the same intercepts as the...
Both of the ellipses have "gaps" at their extreme X values. This is characteristic of most HP 48 conic plots; it is caused by the separate plotting of the two branches.

If A and B have opposite signs, the result is a hyperbola. Try A=1, B=-1, E=1, and F=-1:

Adding a linear term displaces the conic section from the origin. For example, with C=3:
Removing the second-order term in X creates a parabola. Try \( A=0, \ B=1, \ C=1, \ D=E=0, \) and \( F=-1: \)

![Parabola Diagram]

The parabola is turned on end if you interchange X and Y. \( A=1, \ B=0, \ C=0, \ D=1 \) and \( F=-1: \)

![Parabola Turned on End Diagram]

In this case, there is only one branch of the solution for Y.

The interpretation of the resolution parameter, and the effect of the curve-fill flag -31, are the same for conic section plots as they are for function plots. Because \( x \) and \( y \) are treated on a more-or-less equal basis for conic plots, you should generally use the same plotting scale in both directions. AUTO will do this for you automatically; it adjusts the y-scale to be the same as the x-scale, while preserving the y-coordinate of the center of the screen.

### 15.6 Polar Plots

One way to plot an ellipse without the gaps produced by a conic plot is to re-express the defining equation in polar coordinates, then use the polar plot type (POLAR). For example, entering the equation for the first ellipse plotted in the preceding section, \( 'X^2+2*Y^2-9=0' \), make these substitutions:
which yields an equation in $R$ and $\theta$:

$$(R\cos(\theta))^2 + 2(R\sin(\theta))^2 - 9 = 0.$$  

Solve for $R$:

$$R = \sqrt{(18 + 2 + 4\cos^2(\theta) + 4\sin^2(\theta))}/(2\cos^2(\theta) + 4\sin^2(\theta)) \cdot s1$$

Choose the positive branch, and make it the current equation:

$$\{ s1 \ 1 \} \mathbf{t} \text{MATCH} \ \text{DROP} \ \text{STEQ.}$$

Now make a polar plot, with $\theta$ as the independent variable (enter "$\theta$" with $\alpha \rightarrow \beta$):

Polar plots follow much the same logic as function plots, except that the current equation is interpreted to represent the radial coordinate $r$ as a function of the polar angle $\theta$. The resolution parameter (section 15.2.6) determines the increment in $\theta$ between successive plotted points. The default resolution 0 yields an increment of $2^\circ$ in degrees mode, 2 $\text{grad}$ in grads mode, or $\pi/90$ radians in radians mode. The same rules (section 15.4) for interpreting expressions, programs, defining equations ($r = f(\theta)$), and general equations apply to polar and function plots, including the effect of flag $-30$. However, you do need to pay more attention to the independent variable for polar plots:
Since the independent variable is the polar angle rather than the horizontal coordinate, there is no obvious default for the independent variable range. If you don't specify one using INDEP (section 15.2.4), DRAW will plot one cycle: 0 to 360°, 0 to 400 grads, or 0 to 2π, depending on the current angle mode.

- Selecting the polar plot type does not change the default independent variable name, which remains as X. This is convenient for function and conic plots, but less so for polar plots, where you are more likely to want to use a name like θ (\[ \alpha \] \[ \theta \] \[ \Theta \]).

- According to the rules listed in section 15.2.7, LABEL will label the horizontal axis with the independent variable name, which is not a good choice for polar plots. Executing \{ "X" "Y" \} AXES, for example, establishes X and Y as the labels, without affecting the current axes' positions.

The polar plot analog of the special function plot case \( x=constant \) is a current equation of the form \( \theta=constant \), where \( \theta \) is the independent variable. In that case, with flag -30 clear, DRAW plots a straight line through the origin, at an angle \( constant \) from the x-axis. If flag -30 is set, the two sides of the equation are plotted separately, producing a circle \( r=constant \) and a spiral \( r=\theta \).

### 15.6.1 Examples

The following examples are plotted in radians mode, with the plot ranges and current equations as indicated.

#### Figure 8

EQ: \( R='ABS(3*\text{SIN}(\theta))' \)
INDEP: \( \theta \)
XRNG: \(-6.5\) \(6.5\)
YRNG: \(-3.1\) \(3.2\)
**Flower.**

EQ: \( R = \text{ABS}(3 \cdot \text{SIN}(4 \cdot \theta)) \)

INDEP: \( \theta \)

XRNG: -6.5  6.5

YRNG: -3.1  3.2

---

**Teddy Bear**

EQ: \( R = -2 \cdot \text{COS}(4 \cdot \theta) + \text{SIN}(\theta)^{5} \)

INDEP: \( \theta \)

XRNG: -6.5  6.5

YRNG: -1.55  1.6

Flag -31: set

---

If you are sufficiently imaginative, you can see the figure of a teddy bear peering around the axes.
15.6.2 Varying the Angle Increment

If you plotted the last example, you may have noticed that the spiral appeared to be drawn slowly at first, then faster as the spiral grew. This is because when the radial coordinate is small, the default 2° increment is insufficient to cause consecutive points to fall within different pixels, so that some pixels get plotted more than once. Increasing the angle increment to 10° (1.745 radians), for example, speeds up the plotting dramatically, but at the expense of loss of resolution in the outer parts of the spiral.

The HP 48 does not provide for a variable plot resolution, but it is straightforward to make program replacements for DRAW that implement such a feature. The program VARPOL illustrates one approach. Using VARPOL instead of DRAW:
VARPOL

Variable 80 Polar DRAW

<< RCLF \-> flags
  \{ \#0 \#0 \} PVIEW
  DRAX
  3 \-> CF
  19 \-> SF
  16 \-> SF RAD
  22 \-> SF
  'EQ' \-> RCL
  IF DUP TYPE 6 SAME
  THEN RCL
  END
  EQ-\->
  IF DUP 0 SAME NOT
  THEN SWAP
  END
  DROP
  END
  DROP
  PPAR 3 \-> GET
  IF DUP TYPE 5 ==
  THEN OBJ- DROP
  ELSE 0 \"'2*π'\" \->NUM
  END
  ROT \-> 0
  \{ \}
  \-> SHOW
  0 \"'0'\" 2 \->LIST \textsc{tMatch} DROP
  \"'0'\" DUP \-> STO
  DUP 0 \& \textsc{sq}
  OVER SQ + COLCT \sqrt
  \{ \#0 \#0 \} \textsc{px-c}
  \{ \#1 \#0 \} \textsc{px-c} \textsc{re} \textsc{abs}
  SWAP / .18 \textsc{min}
  \textsc{-r dθ}
  \{ \}
  \textsc{for} \textsc{-num}
  \textsc{0 \textsc{-v2 pixon}}
  \textsc{dθ \textsc{-num}}
  \textsc{step}
  \textsc{>>}
  flags \textsc{stoF}
  \textsc{>>}
  \textsc{>>}

\text{Show the plotting.}
\text{Draw axes.}
\text{Symbolic execution mode.}
\text{Complex \textsc{v2}.}
\text{Set polar mode, radians mode.}
\text{Don't error for r=0.}
\text{Get the current equation.}
\text{Get the right side.}
\text{Get independent variable.}
\text{If it's a list, take it apart.}
\text{Otherwise, plot one cycle.}
\text{Save the independent var. name.}
\text{Show independent variable.}
\text{Replace with local name.}
\text{Save the modified expression.}
\text{Compute the radial coordinate.}
\text{Plot the point.}
\text{Compute the polar increment.}
\text{Increment θ.}
VARPOL computes the derivative of the arc length as a function of \( \theta \), then increments \( \theta \) by the amount necessary to increase the arc length by one pixel width. If \( a \) is the arc length measured along a curve, then the differential increment is related to the polar angle by

\[
\delta a = \frac{\delta \theta}{(r^2 + r'^2)^{\frac{1}{2}}}
\]

where \( r = r(\theta) \) is the radial coordinate and \( r' = dr/d\theta \). This program works only with current equations that are expressions or defining equations, which must have sensible derivatives; it also assumes that angular quantities are expressed in radians.

Notice that the plotting takes less time for small \( \theta \) than when using DRAW. VARPOL generally takes longer than DRAW, because it has to evaluate the derivative of the current equation as well as the object itself, but it will often produce a more satisfactory plot.

### 15.7 Parametric Plots

Since HP48 pixels are addressed by rectangular coordinates, polar plots are actually made by converting the polar coordinates of computed point to rectangular coordinates:

\[
x(\theta) = r \cos \theta \\
y(\theta) = r \sin \theta
\]

\( x \) and \( y \) are thus functions of the independent coordinate \( \theta \). Polar plots are a special case of parametric plots. In the general case, the independent variable is not limited to a polar angle, but may be anything.

The current equation for a parametric plot may be an expression or a program that determines both coordinates at once by returning a complex number object. That is, an expression must have the general form \( 'x(t)+i\cdot y(t)' \), where \( t \) is the independent variable; a program must return an equivalent result. As for polar plots, you must ensure that the independent variable and its range are set appropriately with INDEP. If you do not specify any range, the HP 48 will default to the current \( x \)-range, which is not likely to be a useful choice.

A nice example of a parametric plot is a lissajous figure, produced by driving the \( x \)- and \( y \)-coordinates with sinusoidal functions having different frequencies. Start by executing PARAMETRIC ( \([\text{S Wolfe}] \ [\text{PLOT}] \ [\text{PTYPE}] \ [\text{PARA}] \) ) to select the parametric plot type. Then store the following expression as the current equation:
15.7 Plotting

\[3\sin(3T) + 2\sin(4T) \ast i\]

Set the independent variable with

\[
\{ T \ 0 \ 6.5 \} \ \text{INDEP.}
\]

Then, using the default \(x\)- and \(y\)-ranges, the \textit{DRAW} produces this figure:

Choosing a suitable resolution for plotting presents the same difficulties for parametric curves as it does for polar curves. For example, consider the current equation

\[\text{SIGN}(T) \ast \sqrt{\text{ABS}(T)} + T \ast \sin(T^3) + (\text{SIGN}(T) \ast \sqrt{\text{ABS}(T)} + T \ast \cos(T^3)) \ast i\].

Plotting this expression over the range \(-3 \leq T \leq 3\) (\{ T \ -3 \ 3 \} \ \text{INDEP}), with the default resolution of 0, produces the following plot:

For parametric plots, resolution 0 is taken to mean 130 points equally spaced across the independent variable range. In this example, that resolution is too coarse, especially at the beginning and end of the curve.
Executing .01 RES for a finer resolution yields a smoother curve:

![Plot](image)

This is a more satisfactory picture, but an inordinate amount of time is spent plotting near $T=0$. This suggests the program VARPAR, a variation of DRAW for parametric plots analogous to VARPOL for polar plots. There a few differences between VARPAR and VARPOL:

- VARPAR provides for variable resolution, rather than the one pixel spacing used by VARPOL. You set the resolution by using RES with a binary integer argument, which specifies the approximate pixel spacing between plotted points.

- Because of the variable resolution, VARPAR respects the curve fill flag -31, connecting successive points with straight-line segments if the flag is clear.

- VARPAR uses a maximum spacing of 5% of the independent variable range.

Like VARPOL, VARPAR requires that the current equation is differentiable. To use VARPOL with the current example therefore requires one addition: a derivative for SIGN, which does not have a built-in derivative (see section 18.1.1.1). The derivative of this function is not well-defined at zero, but for the current purpose we can just define the derivative to be zero everywhere:

`'derSIGN(x,dx)=0' DEFINE`

With this definition, executing

```
#3 RES DRAX VARPAR
```

yields the picture shown next (following the program listing):
VARPAR assumes that variables EQ and PPAR exist, that the resolution parameter is a binary integer, and that the independent variable entry is a list that includes a range.
The uniformity of the point spacing is more obvious if you plot with flag \(-31\) set, and omit the axes:

---

### 15.8 Truth Plots

A truth-valued function is one that evaluates to a logical value true or false. In HP 48 terms, this translates to an expression or a program that always returns 1 or 0, usually the result of a test command (section 9.3.1). A truth plot is then a map of a truth-valued function of two arguments. For each pair of coordinates \((x, y)\) on the graph screen, the current equation object is evaluated; if it returns true, the corresponding pixel is turned on. The truth plot type is selected for DRAW by the command TRUTH (in the \texttt{PTYPE} menu).

For example, consider the function \(x^3 + y^2\), mapped to truth values according to whether the function value \(mod\ 4\) is less than 2. To plot the mapping, use

\[
'(X^3+Y^2)\ MOD\ 2<4'
\]

as the current equation, and make a truth plot using the default plot ranges:
Because truth plots involve two variables, you must specify names and ranges for the two variables. The default names are X and Y, and the default ranges are just the horizontal and vertical plot ranges. You can use INDEP (horizontal) and DEPND (vertical) to specify alternate names, and to restrict the plotting ranges to any rectangular region. For example, if you execute

\[-3 \, 3 \ \text{INDEP} \quad -2 \, 2 \ \text{DEPND},\]

then redraw the previous plot, you obtain:

Restricting the variables' ranges is often helpful because truth plots take a long time to generate. A typical function plot might require plotting 131 points, but a truth plot using the full screen ranges requires over 8000 evaluations of the current equation. You can use the resolution parameter to plot fewer columns, but it does not affect the y-increment—every row is always plotted.

Because truth plots take so long, it is convenient to have a means of interrupting the plotting such that you can resume it again later without having to start over. The program ITDRAW provides this capability. It creates a truth plot like DRAW, but if you press any key other than [ATTN] during its execution, the program halts. When you press [CTRL] [CONT], ITDRAW resumes. While the program is suspended, you can perform any normal operations except those that might change the graph screen; if you want to do plotting, you should save the current picture, and restore it before continuing ITDRAW.
Get current equation.

Be sure PPAR and PICT exist.

Create local variables.

Compute number of steps.

x and y increments.

If x is a list, correct limits.

Same treatment for y.

Store local names in global variables.

Show the plotting.

x loop.

y loop.

If eq evaluates to true, then turn on the pixel.

If a key has been pressed, then suspend execution.

Restore the plot view.

Purge the global variables.
15.8 Julia Sets

The Mandelbrot set and other Julia sets provide an excellent illustration of truth plots. Julia sets are computed by the iterative process

\[ z^2 + c \rightarrow z \]

where \( z \) is a point in the complex plane, and \( c \) is a constant. A point \( z \) is in the Julia set for \( c \) if the iteration never diverges. The Mandelbrot set is a variation in which the iteration always starts at \( z = 0 \), and the complex plane represents the constants \( c \). To plot the Mandelbrot set, you can use the following program as the current equation:

```
MAND.EQ Mandelbrot Current Equation EAFB
<< X Y R-C DUP
1 15
START
SQ OVER +
NEXT
SWAP DROP ABS 3 <
>>
```

With MAND.EQ selected as the current equation,

```
-21 CF -3.375 1.625 XRNG -1.192 1.231 YRNG ERASE DRAW
```

produces this picture:

![Mandelbrot Set Plot](image)

Fifteen iterations is a reasonable choice for the amount of detail possible with the plot ranges set to show the entire Mandelbrot set. Smaller scale pictures ("zooms") of regions near the boundary of the Mandelbrot set contain more interesting patterns, but
more iterations per point are necessary to resolve the set to an acceptable level of detail. Choosing Julia set constants \( c \) from values at the Mandelbrot set boundary also produces attractive Julia set patterns. The following program JULIA is a generalization of MAND.EQ that can be used to plot either Julia sets or Mandelbrot zooms:

<table>
<thead>
<tr>
<th>JULIA</th>
<th>Julia Plot Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 3</td>
<td>level 2</td>
</tr>
<tr>
<td>level 1</td>
<td>level 1</td>
</tr>
<tr>
<td>((c_x,c_y))</td>
<td>((x_0,y_0))</td>
</tr>
<tr>
<td>(N)</td>
<td>(\text{flag})</td>
</tr>
</tbody>
</table>

\[
\ll 1 \text{ SWAP} \\\n\text{START} \\\n\text{SQ \hspace{1em} OVER \hspace{0.5em} +} \\\n\text{NEXT} \\\n\text{SWAP \hspace{1em} DROP \hspace{0.5em} 3 \hspace{0.5em} ABS <} \\\n\rr
\]

Iteration loop. Return true if \(|z| < 3\).

JULIA takes as its arguments: \((c_x,c_y)\), the Julia set constant; \((x_0,y_0)\), the starting point of the iteration; and \(N\), the number of iterations per point. For plots of the Mandelbrot set, \((x_0,y_0)\) is always \((0,0)\), and \((c_x,c_y)\) are the points on the graph screen. For Julia sets, \((c_x,c_y)\) is different for each plot, and \((x_0,y_0)\) are the pixel coordinates. For example, this picture of the Julia set for \(c = (-.11, .65)\) was produced by this sequence:

\[
.34 \text{ DUP SCALE (0,0) CENTR \{X -1.344 1.344\}} \\\n\text{INDEP \{Y\} DEPND ERASE DRAW (or ITDRAW)}
\]

with
15.8 Plotting

\[-0.11, 0.65\] as the current equation.

For plots of regions that include a lot of points that are not in a Julia set, i.e. for which the iteration diverges, it may be faster to use a variation of JULIA that is slower per iteration but which terminates as soon as the iteration begins to diverge:

<table>
<thead>
<tr>
<th>XJULIA</th>
<th>Julia Set Utility with Exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (c_x, c_y) )</td>
<td>( (x_0, y_0) )</td>
</tr>
</tbody>
</table>

\[
<< 1 \text{ SWAP} \text{ IFERR} \text{ START SQ OVER +} \text{ IF DUP ABS 3 >} \text{ THEN 0 DOERR} \text{ END} \text{ NEXT DROP2 1} \text{ THEN DROP2 0} \text{ END} \text{ >>}
\]

Loop parameters.
Error trap for termination.
Do the iteration.
If \(|z| > 3\), terminate the iteration.
Return true.
Terminated, so return false.

Julia set pictures may take several hours to plot; unfortunately, some of the most interesting patterns may take 200 or more iterations per point, which is almost prohibitive in time and battery consumption. However, using the HP 48 compiler tools that Hewlett-Packard has published, it is possible to write an assembly language version of JULIA that is fifteen times as fast as the user-language version. This zoom of the Mandelbrot set near \((-0.7454, 0.1130)\) was computed with 255 iterations per point:
15.9 Scatter Plots

A scatter plot is a plot of individual disconnected points that are derived from measured or computed data. The term scatter plot indicates that the plotted points appear to be scattered about the graph, in contrast to continuous plots, where the points usually follow each other in a regular progression that forms a smooth curve. DRAW makes scatter plots from data stored in the current statistics matrix, when the SCATTER plot type option is selected. Most of the concepts that we have studied in this chapter for continuous plots carry over directly for scatter plots—digitizing, plot scaling and rescaling, etc. Of the various plot parameters stored in PPAR, only the resolution is not relevant for scatter plots.

DRAW takes the data for a scatter plot from the statistics matrix variable ZDAT (section 20.1), which plays a role for statistics analogous to EQ for function plotting and solving. Scatter plots are made from any two of the columns of the matrix (ZDAT must have at least two columns for this purpose). By default, the first column of data corresponds to the horizontal coordinate, and the second column to the vertical. To make different choices, use i XCOL (section 20.3) to select the ith column as the horizontal dimension, and j YCOL to identify the jth column for the vertical. (i j COL selects both columns at once.) The selected coordinates are not independent and dependent as defined for continuous plots, since the points are just coordinate pairs; but we’ll call them x and y as before. The independent and dependent variable entries in PPAR are used only for axis labels, as explained in section 15.2.7.

For example, if your ZDAT looks like this,

\[
\begin{bmatrix}
11 & 12 & 13 \\
21 & 22 & 23 \\
31 & 32 & 33
\end{bmatrix}
\]

then with column 1 selected as the horizontal coordinate, and column 2 as the vertical coordinate, DRAW plots the points (11,12), (21,22), and (31, 32). 3 XCOL 2 YCOL DRAW plots the points (13,12), (23,22), and (33,32).

An example of a scatter plot is given in section 12.11.1.2 to demonstrate the use of a random number generator. Another example is given by plotting the “noisy” logarithmic curve

\[ y_i = 5 \ln x_i + b_i, \]

represented by the data generated in the example in section 20.3.2.1. After creating the
data, you can create a scatter plot using SCATRE (in the third page of the STAT menu; also execute 1 XCOL 2 YCOL if necessary):

SCATRPLOT is a shortcut command that automatically selects the scatter plot type, then executes AUTO to produce a scatter plot. Autoscaling a scatter plot chooses x- and y-ranges according to the minimum and maximum x and y values in the data, so that there will be at least one point plotted at each edge of the screen, and none outside, with the usual addition 15% added at the bottom for the menu labels. For the current data, this view “hides” the y-axis; zooming out by a small factor frames the plot more nicely:

The command SCLΣ does the same autoscaling as AUTO for scatter plots, without doing the actual plotting. This command is not present in any menu; it is provided for HP28 compatibility.
15.9.1 Plotting Curve Fits
When you have data that corresponds to one of the linear or pseudo-linear curve models, you can determine the model coefficients and other properties of the least-squares fit curve directly from the plot environment. The \texttt{\textbf{FCN}} key (see section 15.4.2) computes and draws the curve that represents the least-squares fit of the current model to the scatter plot data:

\begin{center}
\includegraphics[width=0.5\textwidth]{plot_curve_fit.png}
\end{center}

(You may need to reset the resolution to 0 to reproduce this plot, if you have changed it while working some of the examples in previous sections.)

\texttt{\textbf{FCN}} changes the plot type to FUNCTION, and stores the expression for the fit curve in EQ, including the best-fit parameters. Pressing \texttt{\textbf{FCN}} again activates the function menu for further analysis. If you exit from the plot environment and recall the contents of EQ, you will find an algebraic object representing the fit curve; in the current example, the object is

\[ 1.5848798893 + 4.90171452 \times \ln(X) \]

The coefficients are the same as those computed by LR (section 20.3.2) for this data. The difference between the coefficients in this expression and the ideal values of 1 and 5 arises from the random noise included in the data.

15.10 Bar Charts
A bar chart (plot type BAR) represents data values with vertical bars of lengths proportional to the values. This provides a nice visual presentation of data consisting of relatively few points, especially when there is no well-defined mathematical relationship between the points. DRAW plots a single bar for each data point in the current statistics matrix, from left to right in the order in which the points appear in the matrix. The points are taken from the column specified in \texttt{\textbf{\Sigma PAR}} by \texttt{XCOL}. 
For example, the following are the monthly sales figures for one of Larken Publications' books for the period from August, 1990, through June, 1991:

<table>
<thead>
<tr>
<th>Month</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>August</td>
<td>485</td>
</tr>
<tr>
<td>September</td>
<td>74</td>
</tr>
<tr>
<td>October</td>
<td>40</td>
</tr>
<tr>
<td>November</td>
<td>216</td>
</tr>
<tr>
<td>December</td>
<td>55</td>
</tr>
<tr>
<td>January</td>
<td>7</td>
</tr>
<tr>
<td>February</td>
<td>85</td>
</tr>
<tr>
<td>March</td>
<td>246</td>
</tr>
<tr>
<td>April</td>
<td>221</td>
</tr>
<tr>
<td>May</td>
<td>266</td>
</tr>
<tr>
<td>June</td>
<td>22</td>
</tr>
</tbody>
</table>

This data would make a relatively sparse scatter plot, and is better represented as a bar chart:

This picture was created by entering each of the values from the table above, by entering each successive sales number with $\Sigma+$ (section 20.1), then pressing

1  $\text{XCOL}$  $\text{BARPL}$

(in the third page of the $\text{STAT}$ menu). The command BARPLOT provides a convenient means of creating a bar chart directly from the statistics menu. It automatically selects BAR as the plot type, then executes AUTO. The latter command scales bar charts as follows:

- The $x$-range is set from 0 to $n$, where $n$ is the number of data points in $\Sigma$DAT.
• The y-range is set so that the highest bar (most positive data point) just reaches the top of the graph screen. \( y_{\text{min}} \) is chosen to be \( y_{\text{max}} \) less 115% of the distance from the top of the highest bar to the bottom of the lowest bar (most negative). On a default 64-pixel high screen, this just leaves room for the plot environment menu labels at the bottom of the screen.

• The resolution parameter is set to zero, so that the bars have a width of one logical unit. In combination with the x-range choice, this insures that the bars just fill the graph screen in the horizontal direction.

The bars are presented on the screen in the order in which the corresponding data points appear in \( \Sigma \text{DAT} \), so there is only a weak relationship between the bars and the x-coordinate. With the plot parameters chosen by AUTO, the x-coordinate at the right edge of a bar is approximately equal to the bar number. In general, the bars are plotted, starting at the left edge of the screen, next to each other separated by a blank column, with widths determined by the resolution parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bar Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>1 unit</td>
</tr>
<tr>
<td>Real number ( n )</td>
<td>( n ) units</td>
</tr>
<tr>
<td>Binary integer ( #n )</td>
<td>( n ) pixels</td>
</tr>
</tbody>
</table>

### 15.11 Histograms

A histogram is a bar chart made from data after it has been sorted into numerical ranges, where each bar represents the number of data points that fall into a particular range. With HISTOGRAM as the selected plot type, DRAW counts the elements in one column (specified by XCOL) of \( \Sigma \text{DAT} \) that fall into ranges determined by the plot parameters in PPAR. In particular, the horizontal range is divided into \( N \) equal intervals, where

\[
N = \text{CEIL} \left( \frac{x_{\text{max}} - x_{\text{min}}}{\text{resolution}} \right)
\]

DRAW then makes a bar chart of the interval counts, with bars (approximately) as wide as the intervals they represent. As you can see from the formula above, the interval width is equal to the resolution parameter in PPAR. If the resolution does not divide evenly into the horizontal range, the rightmost bar will be narrower than the others. For a default resolution of 0, DRAW divides the horizontal range into 13 intervals, making the bars 10 pixels wide on a standard size graph screen.
For example, we can plot a histogram of the Larken book sales data from the previous section:

\[
0 \ 500 \ \text{XRNG} \ -1 \ 5 \ \text{YRNG} \ 50 \ \text{RES} \ \text{HISTOGRAM} \ \text{ERASE} \ \text{DRAW}.
\]

Here you can see that the monthly orders tend to be of three sizes: fewer than 100, between 200 and 300, and more than 450.

The command HISTPLOT, which appears in the third page of the STAT menu, executes HISTOGRAM to select the histogram plot type, then AUTO to select plot ranges and draw the histogram. AUTO uses the minimum and maximum values in the selected 2DAT column as the x-range, and uses the current resolution parameter to determine the bar widths. The y-range is set with \( y_{\text{max}} \) equal to the number of entries in 2DAT, and \( y_{\text{min}} = -0.15y_{\text{max}} \), leaving room for menu labels. Usually, this makes the bars fairly short, but it does insure that the full vertical extent of the bars will fit on the screen regardless of the actual distribution of the data.

You can make full use of the y range for histograms by computing the histogram data separately, then using DRAW to plot a bar chart from the data. The histogram sorting is done by BINS, so-called because histogram intervals are often colloquially named "bins."

BINS (in the second page of the STAT menu) takes three arguments:

- \( x_{\text{min}} \) (level 3), which specifies the low (most negative) end of the data range that is to be counted;
- \( w \) (level 2), the bin width; and
- \( N \) (level 1), the number of bins.

BINS sorts the elements in one column (selected by XCOL) of 2DAT into \( N+2 \) bins.
For each element $x$ in $\Sigma$DAT, the $n$th bin count $b_n$ is incremented, where

$$n = \text{IP} \left( \frac{x - x_{\text{min}}}{w} \right) + 1 \quad \text{for } x_{\text{min}} \leq x < x_{\text{max}}$$

where $x_{\text{max}} = x_{\text{min}} + N \cdot w$. $b_N$ is also incremented if $x = x_{\text{max}}$. BINS returns the bin counts to level 2 as a one-column, $N$-element matrix:

$$[[b_1] \ [b_2] \ ... \ [b_N]]$$

It also returns a two element vector $[b_< \ b>]$ to level 1, where $b<$ is the count of data points $x < x_{\text{min}}$, and $b>$ is the count of data points $x > x_{\text{max}}$. For example, with the book sales data still in $\Sigma$DAT,

$0 \ 50 \ 10 \ \text{BINS} \ [3 \ 3 \ 0 \ 0 \ 3 \ 1 \ 0 \ 0 \ 0 \ 1] \ [0 \ 0]$ 

You can observe that the maximum bin count is 3, so a $y$ range from $-.45 \ (3 \times .15)$ to $+3$ is appropriate. Thus, if you store the level 2 matrix in $\Sigma$DAT, and execute

$0 \ 500 \ \text{XRNG} - .45 \ 3 \ \text{YRNG} \ 50 \ \text{RES} \ \text{BAR} \ \text{ERASE} \ \text{DRAW}$,

you obtain this picture:
16. Symbolic Objects and Solutions

The HP 48 and its predecessor the HP 28 are unique among calculators in their ability to apply mathematical operations to “symbolic” quantities—ones for which no numerical value has been assigned. In these calculators, the quantities are represented by objects, specifically names and algebraic objects.

In section 3.5, we discussed the procedure-class nature of algebraic objects, and their similarities and differences with program objects. The name algebraic object is not a particularly good one, since the objects may represent a wider variety of expressions than a strict use of the mathematical term algebraic might suggest. The object name originated in the early days of the HP28C design, when there was still considerable debate among calculator users over the relative merits of RPN calculators and so-called “algebraic” calculators. The HP28 was intended to end that debate by combining the strengths of both calculator types into a unified interface; algebraic objects were so named to convey that the HP28 could deal with objects formulated with a syntax familiar to users of algebraic-style calculators.

In subsequent discussions, we will favor the term symbolic object, which includes algebraic objects and global and local names. Name objects can often be considered as representing elementary expressions, and are usable in most contexts where algebraic objects are allowed.

16.1 Motivations

If you’re a student learning algebra or calculus, or using their techniques in other mathematical or scientific studies, the HP 48’s symbolic capabilities may be very exciting to you. However, if you’re not directly interested in algebra for its own sake, you might wonder why symbolic capability in a calculator is important to you.

The concept of “multiple views” is popular among educators in mathematics, referring to different representations of a function. Each type of view provides different insights into the nature of a function. One view is numerical, describing a function in terms of a table of its values for various arguments. This is the only representation of a function that is possible with a simple numerical calculator. In recent years, graphing calculators have offered a second view—function plots that show functions’ behavior over extended argument ranges. The HP 28 and the HP 48, however, remain the only calculators that offer the third view—the symbolic representation of a function as a mathematical expression—plus the ability to manipulate the expression itself.

Actually, if you use a programmable calculator at all for more than simple keyboard
arithmetic, you are performing a kind of symbolic operation. Any time you perform a calculation more than once, using varying data, you probably represent the calculation symbolically at some point. In particular, when you write a program to automate the calculation, that program is a symbolic operation. You write it to accept certain inputs, without specifying their values, and to compute an unknown result. This is no different in principle from writing a mathematical expression on paper. An expression also "works" with unspecified inputs (variables) and returns a previously unknown value when you evaluate it.

So, in the sense that any program is a symbolic calculation, any programmable calculator is a "symbolic" machine. The major contribution of the RPL-based calculators is that they allow you to apply mathematical operations to the programs themselves, and obtain new programs as results. For example, consider a program that recalls the value of a variable and doubles it. In a BASIC-language computer, for example, the program might look like this, where the "variable" is named X:

\[
\begin{align*}
100 & \text{ Y} = 2 \times \text{X} \\
200 & \text{ END}
\end{align*}
\]

But suppose that after entering the program you realize that you are really interested in the sine of the result, \(\sin(2x)\). In BASIC, you must modify the program by editing it: find the correct program line and enter the \(\text{SIN}\) in the right place, being sure to include parentheses.

On the HP 48, the original "program" consists of the symbolic object '2*X'. To change this into the new program 'SIN(2*X)', all you have to do is execute SIN when the original expression is in level 1. The parentheses are automatically inserted. In effect, the calculator writes a new program for you--all you have to do is use the same keystrokes on the symbolic "program" as you would use with a numerical quantity.

Another way to see the value of the HP 48 capabilities is to consider a general problem-solving process that consists of these steps:

1. Identify the problem.
2. Determine the known and unknown quantities.
3. Figure out the mathematical relationships between the quantities.
4. Solve the relationships for the unknowns in terms of the knowns.
5. For each set of known quantities, evaluate the solved relationships to obtain numerical values for the unknowns.
When you use a conventional calculator, the calculator can only enter the process at the final stage. Once you have equations for the unknowns, you can program those equations into the calculator, enter numerical values for the known variables, and run the programs to return the numerical values for the unknowns. The HP 48, on the other hand, can enter the process as early as step 2. You can use its symbolic capabilities to work out the relationships and solve for the unknowns--steps for which you would need pencil and paper when using another calculator. The symbolic solution that you find with the HP 48 is also the “program” you can use for repeated evaluation of the unknowns with different inputs. Even if the equations you derive can not be solved in closed form for the unknowns, you can still plot the equations or use HP Solve to obtain numerical results, without any further programming.

As an example of this process, consider the classic introductory calculus problem:

\[ A \text{ farmer has 100 yards of fencing to enclose a rectangular field, which is bounded on one side by a river. What length (L) and width (W) of the field gives the maximum area?} \]

**Solution:**

<table>
<thead>
<tr>
<th>Steps</th>
<th>Keystrokes</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The length of the fence is 100 yards.</td>
<td>'L + 2*W = 100_yd'</td>
<td>'L + 2*W = 100_yd'</td>
</tr>
<tr>
<td>2. Solve for L.</td>
<td>'L' (\leftarrow) ALGEBRA (\rightarrow) ISOLVE</td>
<td>'L = 100_yd - 2*W'</td>
</tr>
<tr>
<td>3. Assign this value to L.</td>
<td>'DEF'</td>
<td>'L = 100_yd - 2*W'</td>
</tr>
<tr>
<td>4. The area of the field is L times W.</td>
<td>'L*W = AREA' (\rightarrow) ENTER</td>
<td>'L*W = AREA'</td>
</tr>
<tr>
<td>5. Substitute for L.</td>
<td>'EVAL'</td>
<td>'(100_yd - 2*W) *W = AREA'</td>
</tr>
<tr>
<td>6. To find the maximum area, differentiate the expression.</td>
<td>'W' (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow)</td>
<td>'- (2<em>W) + (100_yd - 2</em>W) = 0'</td>
</tr>
<tr>
<td>7. Collect terms.</td>
<td>'COLCT'</td>
<td>'100_yd - 4*W = 0'</td>
</tr>
<tr>
<td>8. Solve for W.</td>
<td>'W' (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow) (\rightarrow)</td>
<td>'W = 25_yd'</td>
</tr>
</tbody>
</table>
16.1 Symbolic Objects and Solutions

9. Assign this value to \( W \) and evaluate \( L \). 

\[
\text{\texttt{25_def L Eval \textasciitilde 50_yd}}
\]

Answer: The width of the field should be 25 yards, and the length 50 yards.

You can use the HP 48 to formulate and solve the entire problem. With a conventional calculator, all you can do is evaluate the final (dimensionless) answer after you have worked it out on paper, keeping track of the units yourself.

Another example is given by the program SIMEQ (section 12.11.3), which solves a system of simultaneous linear equations. Many other calculators provide this capability either through built-in commands or as program applications. However, without exception (including the HP 48's own built-in method using matrices and vectors), these require you to enter the coefficients and constants rather than the equations themselves. In other words, you must to do the work yourself of inspecting the equations, collecting terms and rearranging if necessary, to determine the coefficients and constants. The SIMEQ program lets you enter the equations in any order, and without having to structure the individual equations in any particular way. It is the HP 48's ability to deal with expressions and equations as data to be manipulated—as symbolic objects—that makes it possible for you to write a program like SIMEQ in a straightforward, compact manner. In other calculator languages, writing a program like SIMEQ would require considerable ingenuity, and would likely end up being harder to use than the usual method of entering coefficients in order.

Some people express disappointment in the HP 48's limited symbolic manipulation facilities—rearranging and collecting terms, solving equations, integration and the like—especially when compared to the features of computer-based algebra systems. The HP 48's capabilities are nevertheless useful, as we will discuss in this and the next chapter, but symbolic manipulation is not the design center of the HP 48. The calculator is oriented towards practical engineering/scientific computation, with a underlying foundation of floating-point arithmetic. Its manipulation operations are intended as an adjunct to the numerical computation. The prime focus of the HP 48's symbolic processing is to provide more power and flexibility than are possible with a numeric-only calculator—the SIMEQ program is a prime example.

The RPL language itself is fully as capable as a basis for computer algebra as LISP or the other similar languages that are commonly used in computer algebra systems. The particular algebra feature set that is implemented in the HP 48 was chosen with regard to practicality on a device with modest processor speed and limited memory. At this writing, no other popular computer algebra system can execute from ROM, and none could possibly execute with only a few kilobytes of free RAM, as the HP 48 commonly does.
The HP 48 takes a generally conservative approach to the execution of symbolic operations. This means that it does not attempt to make decisions for you, but allows you to direct a symbolic calculation at each step. When you enter or compute an expression, the HP 48 does not force the expression into any particular form, but provides expression manipulation commands so that you can rearrange it if necessary. For example,

\[ 'A+B' \ 'B' + \rightarrow 'A+B+B' \]

The HP 48 does not automatically collect terms to return \( 'A+2\times B' \); if you want that result, you can execute COLCT (section 17.1.1).

Another example of the conservative design is in the symbolic solutions returned by ISOL and QUAD (section 16.4). These commands return expressions representing all solutions to their arguments, not just one solution chosen for its “simplicity” or “familiarity.” The solutions are structured so that you can choose the solution or solutions that you want.

The HP 48 chooses a conservative approach for several reasons:

- The calculator can not know what you want. The factors that determine a choice of expression form, or of one solution from among many, are usually not contained in the expression itself but come from external considerations.
- There is no “standard” form for expressions.
- Solutions computed by the calculator should be general and should never obscure any possible solution.
- In a finite-precision, floating-point calculation, the order of operations is important. Two formally equivalent expressions, such as \( '(A+B)+C' \) and \( '(A+C)+B' \), may give quite different results when evaluated numerically (see the discussion of expression structure in section 3.5.2.1). When you set up an expression in a manner that takes this point into account, you do not want the calculator to rearrange the expression.
- Symbolic operations often require a large number of individual steps. If the calculator attempted to standardize the result of each step, it would slow down the overall process.

This approach to symbolic operations means that you will often obtain results that don’t “look” like you expect, or which you have to take extra steps to rearrange.
16.2 General Symbolic Problem Solving

In the preceding section, we outlined a five-step general problem solving process. Now we will review those steps, and see in more detail how they are realized on the HP 48. To illustrate the procedure, we will solve the following problem:

\[ \text{Dad is 40 years old, Son is 10. In how many years will Dad be twice as old as Son?} \]

1. Identify the problem.
Sometimes it's helpful to restate a problem in a more general way, by using variables even for values that are already known. For example:

\[ \text{Dad is } D \text{ years old, Son is } S. \text{ In how many years } T \text{ will Dad be } N \text{ times as old as Son?} \]

This allows you to solve the problem logically once, then enter various choices for \( D \) and \( S \), and find a value of \( N \) for each set of choices.

2. Determine the known and unknown quantities.
The known quantities are the input parameters for the problem--these might be a single value, or a set of input data, or several sets. The corresponding variables are called \textit{known} variables. \textit{Unknown} variables represent the quantities you are going to calculate. In the example, the known variables are \( D \), \( S \), and \( N \). The single unknown is represented by the variable \( T \).

Keep in mind that the choice of which variables are known and which are unknown, is often arbitrary. In the example, you can specify \( T \) and solve for \( N \), rather than the reverse as the problem was originally stated.

3. Figure out the mathematical relationships between the quantities.
This step consists primarily of converting the verbal or conceptual statement of the problem into one or more mathematical relationships. Working the example on the HP 48 proceeds as follows:

<table>
<thead>
<tr>
<th>Verbal Statement</th>
<th>Keystrokes</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Purge existing variables.)</td>
<td>{ D T N } PURGE</td>
<td></td>
</tr>
<tr>
<td>Dad is D years old now.</td>
<td>D ENTER</td>
<td>1: 'D'</td>
</tr>
<tr>
<td>T years later, he'll be...</td>
<td>T +</td>
<td>1: 'D+T'</td>
</tr>
</tbody>
</table>

-520-
Similarly, Son is $S+T$ years old.

Dad is to be $N$ times as old:

The equation in level 1 is the relationship you need.

4. Solve the relationships for the unknowns.

In this step, you apply standard rules of algebra to rearrange the equation such that the unknown variable is isolated as a single quantity on one side. The command ISOL (section 16.4.1) will do this for you automatically if the variable appears just once in the expression or equation. An automatic solution is also possible for general linear or quadratic equations, for which QUAD (section 16.4.3) will find one or two solutions regardless of the specific form of the equation. Since the current example is linear in $T$, QUAD can solve it:

\[ D + T = (S + T) \cdot N \]

\[ T = \frac{D - S \cdot N}{1 - N} \]

It is also instructive to solve the equation using rearrangement methods, to illustrate the process when neither ISOL and QUAD is adequate. The steps on the HP 48 are essentially the same that you might use to solve the problem on paper:

<table>
<thead>
<tr>
<th>Verbal Statement</th>
<th>Keystrokes</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter the expression</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( D + T = (S + T) \cdot N \) | 1: \( D + T = (S + T) \cdot N \) |
| Expand the product. | \( \left\langle \textrm{ALGEBRA} \right\rangle \text{EXP} \) | 1: \( D + T = S \cdot N + T \cdot N \) |
| Subtract $T$ from both sides. | \( T \) \( \left\langle \text{ENTER} \right\rangle \) \( \left\langle \text{—} \right\rangle \) | 1: \( D + T - T = S \cdot N + T \cdot N - T \) |
| Simplify. | \( \left\langle \text{COLCT} \right\rangle \) | 1: \( D = S \cdot N + T \cdot N - T \) |
| Subtract $S \cdot N$ from both sides. | \( S \cdot N \) \( \left\langle \text{ENTER} \right\rangle \) \( \left\langle \text{—} \right\rangle \) | 1: \( -(S \cdot N) + D = T \cdot N - T \) |

At this point all terms containing $T$ are on the right side of the equation, but since there are two such terms, you still can’t use ISOL. First, you must use operations in the EquationWriter subexpression mode to merge the two terms. To activate this mode, press \( \left\langle \mathbf{\wedge} \right\rangle \left\langle \mathbf{\wedge} \right\rangle \) (if you don’t see a menu, press \( \left\langle \text{ATTN} \right\rangle \)):
We want to merge the arguments of the rightmost subtract function by factoring out the common $T$. To do this, the two arguments must have a similar form. With the object cursor on the last $T$, press `RULES` and `=` to:

$$-(S \cdot N) + D = T \cdot N - T$$

Now select the subtract object by pressing `<` twice. Then press `RULES` to activate the operations menu for subtraction:

$$-(S \cdot N) + D = T \cdot N - T \cdot 1$$

The common factor $T$ is on the left in both of the products, so press `-M` to `merge left`:
Now $T$ occurs only once, so the equation can be solved by \texttt{ISOL}. Exit from the EquationWriter environment by pressing \texttt{ENTER}, then

$$T \text{ ISOL} \equiv 'T=\frac{-(S\cdot N)+D}{(N-1)}$$

This is formally equivalent to the result obtained previously with \texttt{QUAD}, differing only in the order of some of the terms.

There are, of course, many problems for which it is impossible in principle to obtain a closed-form symbolic solution for a variable. \(\sin(x) + x = y\), for example, can not be solved for \(x\). This is where HP Solve is invaluable. If you are willing to forgo a symbolic solution, you can skip the current step in the problem-solving entirely, and use the solver menu to obtain numerical solutions for any equation, no matter how many times the unknown variable occurs.

5. \textit{For each choice of known quantities, evaluate the solved relationships to obtain numerical values for the unknowns.}

Once you have an expression or equation that represents a solution to a problem, the only remaining step is to assign specific numerical values to the independent variables, and evaluate the solution object to obtain the corresponding values for the unknown. You can do this in two general ways on the HP 48:

- Assign values to the independent variables using \texttt{STO}, and evaluate the algebraic object that represents the solution. This causes substitution of the numerical values for the variable names, yielding a numerical value for the unknown. In the current example, you can assign values to \(D\), \(S\), and \(N\) from the original problem:

\[40 \ 'D' \ \texttt{STO} \ 10 \ 'S' \ \texttt{STO} \ 2 \ 'N' \ \texttt{STO}\]

Then, with the solution \(T=\frac{-(S\cdot N)+D}{(-1+N)}\) still in level 1,
16.2 Symbolic Objects and Solutions

The result shows that in 20 years, when Dad is 40+20=60, and Son is 10+20=30, Dad will be 2 times as old as Son. You can easily change values to obtain another result—for example, if you give the value 3 to N, and reevaluate the expression, you find that in 5 years, Dad (45) will be 3 times as old as Son (15). Note that if you want to evaluate the expression several times, you need to make copies of it, or to store it in a variable.

• Use HP Solve (Chapter 14). Make your solution object the current equation by pressing [SOLVE] [STEQ], and press [SOLVR] to activate the Solver menu. Then,

\[ 40 \Delta D = 10 \Delta S = 2 \Delta N \underline{=} T \underline{=} 20. \]

The solver menu makes it easy to solve again using different parameters. For N=3:

\[ 3 \underline{=} N \underline{=} T \underline{=} 5. \]

16.3 Symbolic vs. Numerical Solutions

The following are reasons why a symbolic solution is desirable for almost any problem, and preferable to the purely numerical answers that are provided by conventional calculators:

- A symbolic solution is a “global” solution. You can study the behavior of a problem over a range of inputs, just by looking at the mathematical form of the solution.

- A symbolic solution acts as a “program” that allows you to determine numerical results at any time. Once you have the symbolic solution, you can assign values to the variables and evaluate the symbolic object to obtain specific numerical results.

- Even if you're using HP Solve for purely numerical answers, it is faster when you want a series of results to rearrange the current equation symbolically so that you can use [EXPR=] rather than solving numerically each time.

- A symbolic expression tells you something about the calculation “history” and parameters that have contributed to an answer. Once you convert an expression to a number, you wipe out the logical trail that led to the number.

To solve a problem symbolically means to take the equations that represent the problem and rearrange them using the rules of algebra until you manage to isolate the unknown variable's name. (In this discussion, we will use the term equation to refer either to an actual equation or to an expression \( f(x) \) that is understood to represent the equation \( f(x) = 0. \)) To “isolate” an unknown \( x \) means to obtain an equation of the form

-524-
Symbolic Objects and Solutions

\[ x = f(y, z, \cdots), \] where \( x \) does not appear in the right side of the equation, and \( y, z, \) etc. are known quantities.

The HP 48 provides two types of tools to help you obtain symbolic solutions once you have entered the equation(s) for a problem. First, there are several commands for rearranging expressions, which approximate the steps you carry out in pencil-and-paper calculations. Included among these the EquationWriter RULES operations for detailed expression manipulation, EXPAN (expand), for distributing multiplication and powers, COLCT (collect), for combining like terms, and \{MATCH and \{MATCH for making a variety of substitutions. Second, there are two automatic expression/equationsolvers, ISOL and QUAD, which can carry out several steps in the solving process at once.

ISOL and QUAD are certainly the easiest methods of solving for a variable. However, both have certain restrictions in their application: ISOL yields a true \textit{solution} only if the unknown variable's name appears just once in the equation. QUAD permits multiple occurrences of the unknown's name, but a QUAD result is only a solution if the equation is second order (quadratic) or lower in the unknown. For equations that don't fit either of these criteria, the typical HP 48 symbolic solving process is a combination of the two types of solving tools. You use the expression manipulation commands to convert an equation into a form suitable for final solution by ISOL.

16.3.1 \( -Q \) and \( -Q_{\pi} \)

One of the problems associated with carrying out symbolic manipulations with floating-point numbers is the loss of information and precision associated with representing rational fractions with decimal numbers. The HP 48 provides two commands, \( -Q \) and \( -Q_{\pi} \), that can help you reconstruct a fraction from a number that has been converted into decimal form.

\( -Q \), which stands for \textit{to quotient}, attempts to find a rational fraction equivalent of a real number, expressed as an algebraic object. For example,

\begin{verbatim}
3 FIX .494347379239 -Q \rightarrow '43/87'
\end{verbatim}

The number display setting is significant:

\begin{verbatim}
STD .494347379239 -Q \rightarrow '481/973'
\end{verbatim}

\( -Q \) attempts to find a quotient of integers which matches its argument to the number of decimal places specified by the current number display mode. Standard (STD) mode indicates 11 decimal places, and fixed, scientific, and engineering modes specify the
number indicated by the original mode argument, e.g. nine places for 9 FIX, etc. In particular, \( \text{\texttt{\neg Q}} \) finds the fraction with the smallest denominator that matches the argument within the error limit set by the display format. For example,

\[
\text{STD 1.61803398875 } \text{\texttt{\neg Q}} \Rightarrow '514229/317811'.
\]

Notice that the denominator does not have 11 digits. The fraction \( 139583862445/86267571272 \) is another representation of the original argument that evaluates to the same decimal value as \( 514229/317811 \); \( \text{\texttt{\neg Q}} \) chooses the latter because the denominator is smaller.

\( \text{\texttt{\neg Q}} \pi \) ([ALGEBRA] menu, second page) is a variation of \( \text{\texttt{\neg Q}} \) that includes a factor of \( \pi \) in the fraction computation. That is, \( \text{\texttt{\neg Q}} \pi \) finds a quotient for its argument and another for the argument divided by \( \pi \). It returns the quotient with the smaller denominator, multiplied by \( \pi \) if appropriate. Since \( \pi \) itself is irrational, and not precisely represented by a floating point number, \( \text{\texttt{\neg Q}} \pi \) will not necessarily return a result containing \( \pi \) even if you started with a product of (floating-point) \( \pi \):

\[
\pi 2 / \text{\texttt{-NUM 5 FIX \neg Q}} \pi \Rightarrow '1/2*\pi'
\]

\[
\pi 2 / \text{\texttt{-NUM STD \neg Q}} \pi \Rightarrow '573204/364913'
\]

If you want to guarantee that a particular argument converts to a factor of \( \pi \), you should divide the argument by \( \pi \), execute \( \text{\texttt{\neg Q}} \), then multiply by \( \pi \) again:

\[
10 \text{\texttt{\neg Q}} \pi \Rightarrow '1/4*\pi'
\]

16.4 Automated Symbolic Solutions: QUAD and ISOL

The commands QUAD and ISOL automatically carry out several steps toward the symbolic solution of an expression or equation for an unknown. Each operates on an algebraic object in level 2, and a global name specifying the unknown variable in level 1. Each returns an equation representing the solution, with the name of the unknown variable on the left and an expression for the solution on the right. For example, using ISOL to solve an equation for \( X \),

\[
'((A-B)*X)=(-(A*Y)+C)' \quad 'X' \quad \text{ISOL} \quad 'X=-(A*Y)+C)/(A-B)'.
\]

\( X \) appears only on the left side of the result. This does not change the current value of \( X \); if you want to assign the right-side expression as the new value for \( X \), you can execute \text{DEFINE} (section 8.5). Or, if you want to extract the right-side expression itself for
some further calculation, you can take the equation apart using EQ→ (in the first page of the PRG OBJ menu).

16.4.1 ISOL

ISOL solves an equation for an unknown much the same way you would with pencil and paper. It finds the (first) term containing the unknown, and moves it to the left side of the equation, moving all other terms to the right side. Then, if the unknown is contained in the argument of a function, the inverse of the function is applied to both sides of the equation. If the result does not have the unknown by itself on the left, then the whole process is repeated until only the unknown remains on the left side. For example,

'2*X+8=0' 'X' ISOL ☐ 'X=-4',

and

'A+B*X/C=D' 'X' ISOL ☐ 'X=(D-A)*C/B'

ISOL can find a solution for an equation if three conditions are met:

1. The unknown variable name appears just once in the equation.
2. The unknown appears only in the arguments of HP 48 analytic functions.
3. No variables in the equation contain algebraic objects or programs that have the unknown in their definitions.

If either of the first two conditions is not satisfied, ISOL returns the Unable to Isolate error. The second condition is really part of a circular definition--ISOL will only work with analytic functions, but part of the definition of an HP 48 analytic function is that the HP 48 "knows" its inverse and hence can isolate its argument. The object description analytic is loosely derived from the mathematical definition of an analytic function as one that is continuous and differentiable. If you inspect the HP 48 function set, it is usually easy to figure out why a particular function is analytic or not in the HP 48 sense if you keep the mathematical definition in mind. Functions like IP, FP, MOD, or MANT are not continuous, and hence are not classified as analytic. ABS is an example of a function that is continuous but not differentiable--its slope changes abruptly at 0.

16.4.2 SHOW

If the third condition listed for ISOL in the preceding section is not met, no error is reported, but the result may not be meaningful. ISOL only works at the immediately visible level of an expression. That is, it does not execute any of the variable names to substitute their values. For example, if you isolate B in 'A+B=C', the result is 'B=C-A'. But if A has the value 'B-C', then the result ISOL returns does not
represent a proper solution. To prevent such occurrences, you can evaluate the original equation repeatedly until all implicit uses of the unknown are revealed. This may have the unwanted side effect of making other substitutions that you do not want, including any current value of the unknown itself. The correct approach is to use SHOW, with the same arguments as you would use for ISOL. This command makes any substitutions necessary to make references to the specified name explicit in the level two object:

\['A + B = C'  'B' \text{SHOW} \Rightarrow 'B - C + B = C'.\]

You can then proceed to solve the result equation.

SHOW has a second behavior that is complementary to that just described. If instead of entering the second (level 1) argument as a name, you enclose that name in a list, SHOW evaluates all names in the level 2 object except the specified name. The list may actually contain more than one name, in which case none of the listed names are evaluated by SHOW. Any functions with numerical arguments are executed. The primary use of SHOW in this manner is for cases where an algebraic object is to be executed repeatedly; SHOW is used to remove as many functions and symbolic references as possible from an expression to minimize its evaluation time. An example of this application is given in section 14.3.1. Note that symbolic execution mode (flag -3 clear) must be active when SHOW is evaluated with a list argument, to prevent the execution of functions that use the listed names as arguments.

16.4.3 QUAD
QUAD is designed for solving quadratic equations \(ax^2 + bx + c = 0\), where \(x\) is the unknown variable, and \(a\), \(b\), and \(c\) are constants with respect to \(x\). QUAD does not require the equation to have this form. It takes an arbitrary expression or equation and converts it to a second-order polynomial in the specified variable by computing a second-degree MacLaurin polynomial (like TAYLR). This representation is exact if the original expression is first or second order in the variable. QUAD then applies the quadratic formula to the coefficients in the polynomial to obtain its solution.

As part of the process of determining the coefficients, the original symbolic argument is evaluated. Therefore, if you want to prevent substitution for the names in the object, you must purge the corresponding variables before executing QUAD.

In keeping with the HP 48's generally conservative approach to rearranging expressions, QUAD does not attempt to constrain its result into a standard form, so you may have to do some manipulation of the result to make it look like a "textbook" solution. For example, if you solve the standard quadratic equation:
Symbolic Objects and Solutions

\[ A \times X^2 + B \times X + C = 0 \]

\[ X = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \]

the result is clearly not as compact as it might be. You can improve the appearance of the result with COLCT:

\[ COLCT \quad \frac{5 \times (\sqrt{-(4 \times A \times C) + B^2}) \times s1 - B}{A} \]

which is closer to but still not quite the same as the textbook result

\[ \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \]

however, the two forms are equivalent when evaluated. Note that the \( \pm \) is represented in the QUAD result by the variable \( s1 \). This concept is explained in the next section.

When its argument is only first order in the unknown, QUAD returns a single solution:

\[ A \times X + B = 0 \]

\[ X = \frac{-B}{A} \]

It even attempts to solve the zero-order case:

\[ A \times X = 0 \]

\[ X = \frac{-A}{0} \]

QUAD will not work, however, when the unknown variable must contain a unit object for dimensional consistency with the rest of the expression (TAYLR, which shares computing logic with QUAD, has the same problem.) If \( f(x) \) is the argument expression, then QUAD computes

\[ f(0) + x \cdot f'(0) + x^2 f''(0) \]

that is, \( f \) and its derivatives are evaluated at \( x=0 \), not at \( x=0 \_units \), so that the successive terms in the quadratic have inconsistent units. See section 21.3.8.

16.5 Multiple Roots

A fundamental result of algebra is that many expressions or equations, even some quite simple looking, have more than one solution (root). This principle is recognized in the behavior of QUAD and ISOL.
Quadratic equations always have two solutions, which are commonly combined into a single expression with the use of a ± sign. QUAD achieves this combination by returning both solutions as a single expression containing the (global) name s1, which represents a ±. We will call a name of this type an arbitrary sign. The use of a global name in this manner gives you a means of choosing one sign or the other. To choose the positive root, you store +1 in s1 and evaluate the expression. For the negative root, you similarly use -1. You can also leave s1 without a value as long as you like, so that you can perform additional calculations on both roots together.

ISOL also returns a single expression representing all possible solutions for its symbolic object argument. Such solutions may contain one or more arbitrary signs, so ISOL uses the names s1, s2, ... to represent each successive ± required by the solution. ISOL may also include the global names n1, n2, ..., as needed, which represent arbitrary integers. You may assign any integer value 0, ±1, ±2, ..., to each of these names; each combination represents a different solution to the original expression. For example,

\[ \text{RAD 'SIN(X^2)=Y' 'X' ISOL }\xrightarrow{i} \ 'X=s1*√(\text{ASIN}(Y)*(-1)^n1+\pi*n1')\]

Here you can observe one arbitrary sign s1 and one arbitrary integer n1. n1 appears twice in the expression, meaning that the same choice of integer must appear in both places.

The appearance of arbitrary signs and integers may be confusing if you expect to find a solution to a problem. However, it is not ISOL or QUAD that is introducing complexity into your problem; they are just showing you the mathematically complete result, and not trying to choose one particular root as "better" than any other. As a matter of fact, there is no automatic criterion that the commands could use to choose one root over another; that is a choice that only you can make by considering factors of the problem that are separate from the equation being solved.

For example, consider the equation \( x^2 = y^2 \). For any \( y \), there are two values of \( x \) that satisfy the equation, \( x = y \) and \( x = -y \). Mathematically, there is no distinction between the two; either could be the correct choice for a particular physical problem. You might prefer the positive root because it "looks nicer," but such value judgments are not practical for an automated procedure like ISOL. Besides, -\( y \) might be the preferred choice on other grounds.

This problem is obscured somewhat by HP Solve, which only returns one answer at a time. In cases with multiple roots, HP Solve usually (but not always) returns the root that happens to be closest to the value stored in the unknown variable (the "initial guess") when the solving starts. By supplying an initial guess (section 14.5), you are choosing a particular root in advance. If you don't supply an initial guess, you must take
your chances with whatever value was left in the unknown variable by a previous calculation. When the variable doesn't exist, the root-finder uses zero as an initial guess—which may or may not be a good choice for the problem at hand.

The HP 48 does provide a flag-controlled mode called principal value mode, in which a default choice for all arbitrary signs and integers is supplied automatically. When this mode is active (flag −1 clear), arbitrary signs are always chosen to be positive, and arbitrary integers are set to zero. The purpose of this mode is to provide an answer to a problem, perhaps to give you a general idea of the appearance of the answer, without the distraction of the arbitrary constants. However, the results returned by ISOL and QUAD in this mode may not be appropriate at all for a real problem.

For example, consider the equation $x^3 = -1$. You can see by inspection that $x = -1$ is one root; imagine that −1 is the correct choice for a particular problem. Solve this equation for $x$ in principal value mode:

$$-1 \text{ SF } 'X^3=-1' 'X' \text{ ISOL } \Rightarrow 'X=(.500000000001,.866025403784)'.$$  

The complex result is in fact one of the three cube roots of −1, but it is not the one we have specified. Now try solving again, with principal value mode off:

$$-1 \text{ CF } 'X^3=-1' 'X' \text{ ISOL } \Rightarrow 'X=\text{EXP}(2\pi i n1/3)\times(.500000000001,.866025403784)'.$$  

Translated to common notation, this result is

$$x = e^{2\pi in1/3}\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right),$$

where we have replaced the approximate decimal values with fractions. $n_1$ is an arbitrary integer, which means that you can choose any integer value $0, ±1, ±2, \cdots$, for $n_1$ to obtain a cube root of −1. There are only three distinct roots for a cubic equation, which you can obtain with any three consecutive values of $n_1$. Other values of $n_1$ just reproduce the same roots. The following table lists the values returned by the HP 48 along with the exact roots, for $n_1 = 0, 1,$ and 2 (the errors in the last decimal place arise from the inaccuracy of the floating-point representation of $1/3$):
16.5 Symbolic Objects and Solutions

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>HP 48 Result</th>
<th>Exact Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(.5000000000001, .866025403784)</td>
<td>($\frac{1}{2}, \frac{\sqrt{3}}{2}$)</td>
</tr>
<tr>
<td>1</td>
<td>(-1, 4.465E-12)</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>(.5000000000001, -.866025403784)</td>
<td>($\frac{-1}{2}, \frac{-\sqrt{3}}{2}$)</td>
</tr>
</tbody>
</table>

Unless you force it by setting principal value mode, the HP 48 does not attempt to choose one possible root over any other when you use ISOL. There's really no mathematical grounds on which the calculator could make such a choice. As you can see from this example, the "obvious" choice of $n_1 = 0$ does not return the "obvious" answer to the problem, $x = -1$.

You might wonder why ISOL and QUAD use arbitrary integer and sign names in their results, rather than perhaps returning one or more expressions, each of which represents a different root. There are three primary reasons:

1. In general, a problem may have any number of roots, even an infinite number. It is obviously impossible to return an infinite number of objects, and the HP 48 has no way to tell that there is a finite set of different roots among the infinite possibilities represented by one or more arbitrary integers.

2. By returning a single expression to represent a general result, that expression is immediately suitable for use as an argument for further operations, symbolic or numerical. Dealing with even a finite set of multiple results would be very difficult in a program.

3. The use of ordinary names to represent the arbitrary constants allows you to use the normal methods available for variables (STO, the VAR menu, HP Solve, etc.) to select values for the arbitrary signs and integers.

### 16.5.1 Using the Solver Menu to Select Roots

The solver menu (section 14.2) provides a convenient method for selection of individual roots from a multiple root solution provided by ISOL or QUAD. By storing an expression returned by one of these commands as the current equation, you obtain a solver menu containing all of the arbitrary signs and integers, any other variable names in the expression, and the $\underline{EXPR=}$ key. Then you can use the menu keys to select values for the variables, and press $\underline{EXPR=}$ to obtain the evaluated expression.

To illustrate, return to the example $x^3 = -1$. First, if necessary, press $\text{[MODES]}$ $\underline{SYM=}$ $\underline{SYM=}$ to activate symbolic execution mode (make sure the white square appears in the $\underline{SYM=}$ key label). Then:
Symbolic Objects and Solutions

Keystrokes: Results:

[→] MODES NEXT – 1 [⇒] CF

← MODES 3 [⇒] FIX

'X^3 = -1' [⇒] ENTER

'X' [ALGEBRA] [⇒] ISOL

1: 'X = EXP(2*π*i*n1/3) *

(0.500,0.866)'

[PRG] OBJ [⇒] EQ

2:

1: 'X' Split the equation.

1: EXP(2*π*i*n1/3) *

(0.500,0.866)'

← SOLVE [⇒] STEQ [⇒]

SOLVE [⇒] ZSOLVR

Make the result the current equation, and activate the Solver menu.

n1 is the only variable in the current equation. To compute all three roots, use n1 = 0, 1, and 2:

Keystrokes: Results:

0 [⇒] N1 [⇒] EXPR

1:

EXPR: (.500, .866)

1 [⇒] N1 [⇒] EXPR

2:

EXPR: (.500, .866)

1: 'EXPR: EXP(2*π*i*n1/3) *

*(0.500,0.866)'

[→] -NUM

2:

EXPR: (.500, .866)

1: (-1.000, 4.465E−12)

2 [⇒] N1 [⇒] EXPR [⇒] [→] -NUM

3:

EXPR: (0.500,0.866)

2: (-1.000,4.465E−12)

1: (0.500,−0.866)

The stack now contains all three cube roots of −1.
16.6 Algebraic Objects as Programs

In preceding sections we have described expressions or equations as "programs." In most other calculators and computer languages, a program is not a mathematical object—it might contain mathematical expressions, but the program itself is usually a series of numbered lines, each containing one or more instructions. But take away the line numbers, and what you have is just a series of data and instructions that are meant to be executed sequentially and automatically. It is easy to recognize that an HP 48 program works like this, since by definition a program contains a progression of any HP 48 objects that are executed when the program is executed. But it is not so obvious for algebraic objects, since they look like mathematical expressions or equations, which are not commonly thought of as programs (see also section 3.5).

Using the object decomposition tools discussed in Chapter 3 and elsewhere, it is a straightforward matter to convert algebraic objects into RPN sequences, and vice-versa. The program -RPN listed below automates this process, converting any algebraic object into a list, in which the objects that originally constituted the algebraic object's definition are presented in the equivalent RPN order.

<table>
<thead>
<tr>
<th>-RPN Convert to RPN</th>
<th>7447</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>level 1</strong></td>
<td><strong>'expression'</strong></td>
</tr>
<tr>
<td><strong>&lt; OBJ~</strong></td>
<td><strong>IF OVER</strong></td>
</tr>
<tr>
<td><strong>THEN - n f</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>&lt; 1 n</strong></td>
</tr>
<tr>
<td></td>
<td><strong>FOR i</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ELSE 1 -LIST SWAP DROP</strong></td>
</tr>
<tr>
<td></td>
<td><strong>END</strong></td>
</tr>
<tr>
<td></td>
<td><strong>&gt;&gt;</strong></td>
</tr>
</tbody>
</table>
An example of using →RPN:

\[ 'A+2*\text{SIN}(B/C)' \rightarrow \text{RPN} \rightarrow \{ A \ 2 \ B \ C / \text{SIN} \ * \ + \} \]

### 16.7 Refining User-Defined Functions

An important constituent of HP 48 symbolic operations is the set of functions (section 3.1) that you can use within symbolic objects' definitions. The HP 48 provides an extensive set of real- and complex-valued functions; by installing libraries (section 3.4.11), you can add even more. In section 8.5, we describe the construction of so-called *user-defined functions*, which you can use to add to the built-in and library function set. User-defined functions on the HP 28 were subject to certain limitations compared with built-in functions; the HP 48 has added three commands to remove these limitations.

#### 16.7.1 Preventing Evaluation: QUOTE

In the course of the ordinary evaluation of an algebraic object, each function's arguments are evaluated before the function itself is executed. There are certain functions, however, that have the ability to prevent the evaluation of one or more of their arguments. \( \partial \) is a good example; neither the name representing the variable of differentiation nor the expression to be differentiated should be executed prior to \( \partial \) itself. When you enter an expression such as \( \partial X(\text{SIN}(X)) \), the HP 48 automatically (and invisibly) enters the \( X \) and the \( \text{SIN}(X) \) as full algebraic objects themselves. That is, the overall expression is equivalent to the RPN sequence

\[ '\text{X}' \ '\text{SIN}(X)' \ \partial. \]

The first two objects are "quoted" (section 3.8) as algebraic objects so that execution leaves them unchanged as the sequence is evaluated. For most functions, the internal quoting is not done, e.g. \( \text{MAX}(X,\text{SIN}(X))' \) translates to the sequence

\[ X \ X \ \text{SIN} \ \text{MAX}. \]

User-defined functions do not intrinsically have this ability to quote their arguments. If you evaluate \( \text{f}(x,y,z)' \), where \( f \) is the name of a user-defined function, \( x \), \( y \), and \( z \) are evaluated before \( f \). To correct this deficiency, the HP 48 provides the function QUOTE, the execution of which simply returns its single argument unevaluated. Thus the evaluation of \( \text{f}(\text{QUOTE}(x),y,z)' \) executes \( y \) and \( z \) as usual before \( f \), but leaves \( x \) unchanged for \( f \).

As an example, suppose you want to make the polynomializer program POLY (section 17.4) into a user-defined function. Rather than rewriting POLY, it can be incorporated
into another program POLYF:

\[
\begin{array}{|c|c|c|}
\hline
\text{POLYF} & \text{Polynomializing Function} & \text{CEC9} \\
\hline
\text{level 2} & \text{level 1} & | & \text{level 1} \\
\hline
\langle \langle f(x) \rangle \rangle & \langle \langle x \rangle \rangle & \text{POLY} & \langle \langle P_N(x) \rangle \rangle \\
\hline
\end{array}
\]

Evaluating ‘POLYF((X+1)^3,X)’, for example, will expand the cubic polynomial, but the result will not contain X if that variable has a value. To keep X in the result regardless of any value X might have, you can quote the arguments:

‘POLYF(QUOTE((X+1)^3),QUOTE(X))’ EVAL  \( \equiv \) ‘X^3+3*X^2+3*X+1’.

16.7.2 Applying Functions without Evaluation: APPLY

Another capability of certain built-in functions is to test their arguments for special cases for which the functions are to be returned unexecuted. For example, when SIN is applied to a symbolic argument, it just returns itself unexecuted, unless the argument is \( \pi \), in which case it returns zero. The function APPLY extends this capability to user-defined functions.

APPLY is used in expressions with the syntax

‘APPLY(name, expression, ..., expression)’.

When this is evaluated, each of the argument expressions is evaluated, and then the unevaluated name is appended to the result as if the name signified a user-defined function. For example,

‘APPLY(F,1+1,X)’ EVAL  \( \equiv \) ‘F(2,X)’

To understand the use of APPLY, consider the execution of a typical user-defined function with a symbolic argument. Create a user-defined function for the cosecant:

‘CSC(X)=INV(SIN(X))’ DEFINE
Then, if \( Z \) is undefined,

\[
\text{\texttt{\textasciitilde CSC(Z)} EVAL \rightarrow \text{\textasciitilde INV(SIN(Z))}}.
\]

Compare this with the result of evaluating \('\text{SIN(Z)}'\), which just returns the same expression unchanged. The following version of \( \text{CSC} \) uses \( \text{APPLY} \) for symbolic arguments:

\[\begin{array}{|c|c|}
\hline
\text{CSC} & \text{Cosecant Function} \\
\hline
\text{level 1} & \text{level 1} \\
\hline
x & \text{csc}(x) \\
\text{\textasciitilde symb} & \text{\textasciitilde CSC(symb)} \\
\hline
\end{array}\]

CSC executes \( \text{APPLY} \) as an RPN command. Used this way, \( \text{APPLY} \) expects two arguments: a global or local name in level 1 (the function name) and a list containing the arguments to which the function is to be applied.

\[
\{ \text{expr}_1, \ldots, \text{expr}_n \} \text{\textasciitilde name \texttt{APPLY} \rightarrow \text{\textasciitilde name(\text{expr}_1, \ldots, \text{expr}_n)}}.
\]

The apparently pointless sequence \('\texttt{\textasciitilde symb} \texttt{SHOW}'\) in CSC has a definite purpose. Unfortunately, all versions of the HP48 (A-E) at the time of this writing have a defect in \( \text{APPLY} \) such that all of the expression arguments of its results are quoted in the sense described in section 16.7.1. This means that such results can not be evaluated further, since the internal quoting prevents the argument expressions from being evaluated. Without the \texttt{SHOW} you would find, for example,

\[
\text{DEG 30 'Z' STO 'Z' CSC EVAL \rightarrow 2,}
\]

but then

\[
\text{'Z+Z' CSC EVAL \rightarrow 'CSC(Z+Z)'}
\]
The last result would remain unchanged even after repeated uses of EVAL. During the execution of SHOW, the result is taken apart and then reassembled correctly, so that

\[ 'Z+Z' \text{ CSC EVAL} \rightarrow 1.15470053838 \]

The SHOW slows execution a little, but the result is more generally useful. You can substitute any other name for \( t \), although it is best to use an uncommon one that you are not likely to use for a variable (section 18.1.2). Future versions of the HP 48 may correct this problem, in which case the SHOW would be superfluous.

The definition of CSC above uses a program its defining procedure (section 8.5), so that a derivative is not automatically supplied. But it is straightforward to define a derivative for CSC (see section 18.1.1.1)

\[ '\text{derCSC}(x, dx) = \frac{\text{CSC}(x)}{\text{TAN}(x)} \times dx' \text{ DEFINE} \]

Then

\[ '\text{CSC}(Y^2)' \quad \text{\( Y \)} \quad \partial \rightarrow '\frac{\text{CSC}(Y^2)}{\text{TAN}(Y^2)} \times (2*Y)' \]

16.7.3 Preserving Local Variables' Values:

If a user-defined function contains any functions that delay evaluation of their arguments, such as \( \partial \) or QUOTE, the local names associated with the user-defined function may be present in the function's result. Since the corresponding local variables are discarded after execution of the function, further evaluation of the result could result in the Undefined Local Name error. To forestall this problem, user-defined functions defined with expressions automatically inspect their results for the presence of any of the function's local names. If any are found, the result is returned as an argument of the \( \text{where} \text{ function} \).

For example, consider the following user-defined function:

\[ 'F(x) = \partial x(\text{SIN}(x))' \text{ DEFINE} \]

Since \( \partial \) in an expression executes in a step-wise manner (section 18.1), it and the local name \( x \) will be present in any result of evaluating \( F \), e.g.

\[ 'F(Z)' \text{ EVAL} \rightarrow '\text{COS} (x) \times \partial x(x) \mid (x = Z)' \]

This result is more legible in the EquationWriter:
You can read the expression starting at the vertical bar as "where x equals Z." That is, the subexpression to the left of the bar is to interpreted normally, with the understanding that ultimately the value Z is to be substituted for the symbol x. This is just what | does when it is executed; it evaluates its first argument, repeatedly if necessary while substituting for the names as indicated in the remaining arguments, until the names no longer appear. Thus, evaluating the previous result:

EVAL \(\rightarrow\) 'COS(Z)'

The syntax of the where construct provides for any number of variable assignments:

'expression \( (x_1, \ldots, x_n) \mid (x_1 = value_1, \ldots, x_n = value_n)\)'.

Thus with FXYZ defined by

'FXYZ(x,y,z)=QUOTE((x+y+2)/(x*y*z))' DEFINE,

then

'FXYZ(1,2,3)' EVAL \(\rightarrow\) '(x+y+2)/(x*y*z)\mid (z=3,y=2,x=1).

Notice that there are no parentheses around the subexpression to the left of the | (assuming that flag −53 is clear), even though that subexpression is an argument of |. | has the lowest precedence of any function. In an EquationWriter picture, the extent of its argument is shown by the bar's vertical size:
The bar extends upwards to the top of the left argument, and downwards to the bottom of the argument, plus enough additional to span the variable assignments on its right.
17. Expression Manipulations

The symbolic solution commands ISOL and QUAD are easy to use, but fairly limited in the range of problems they can address. The HP48 therefore provides a number of tools for performing symbolic manipulations on expressions and equations. These tools may be used to rearrange a symbolic object into a form suitable for ISOL or QUAD, or for any of a number of other purposes.

17.1 Extensive Manipulations

The most common operations you might perform in the course of finding symbolic solutions are these:

- **Reordering terms**: changing the order of sums and factors. For example, you might move all of the terms containing an unknown variable together so that multiple occurrences of the unknown can be merged into a single occurrence. This is achieved in the HP 48 by means of the various RULES operations in the Equation-Writer subexpression mode, especially the term-moving operations «<T and T->, and the association operations A→ and ←A. Also, to move one term from one side of an equation to the other, you can subtract the term from both sides using the ordinary − command.

- **Expansion**: distribution of products or powers over sums. The conversion of \(a \times (b + c)\) into \(a \times b + a \times c\) is an example of the distribution of a product over a sum. An example of the distribution of a power is the expansion of \((a + b)^2\) into \(a^2 + 2ab + b^2\). Expansion in the HP48 is represented by EXPAN, and by the RULES operations D→ and ←D.

- **Merging terms**: Once you have all of the terms containing an unknown gathered together, the next step is usually to combine as many of these terms as you can, to minimize the number of occurrences of the unknown—to a single occurrence, if possible. The principal tools for this purpose are COLCT and the RULES merge operations M→ and ←M.

As an example of HP48 symbolic manipulation, consider solving the equation \(a(x + y) = bx + c\) for \(x\).

1. Press \([\downarrow]\[\text{EQUATION}\]", and enter the equation:
2. Distribute the term $A \cdot (X+Y)$:

3. Move the term $A \cdot Y$ to the right side:
4. Move $B \cdot X$ to the left:

\[ \text{nine times (select the =)} \]

\[
\begin{align*}
A \cdot X & = B \cdot X = C \cdot Y \cdot A \\
\end{align*}
\]

5. Now merge the terms that contain $X$:

\[
\begin{align*}
(A-B) \cdot X & = C \cdot Y \cdot A \\
\end{align*}
\]

6. Commute the factors of the first term:

\[
\begin{align*}
X \cdot (A-B) & = C \cdot Y \cdot A \\
\end{align*}
\]
7. Isolate $X$: \[ \text{four times (select the =)} \]

\[
X = \frac{(C-Y\cdot A)}{-B+1}
\]

Now return the object $X = (C-Y\cdot A)/(-B+A)$, in which $X$ has been isolated on the left side of the equation.

This result can also be obtained by using programmable commands:

- **Distribute:**
  \[
  A \cdot (X+Y) = B \cdot X + C
  \]
  EXPAN
  \[
  A \cdot X + A \cdot Y = B \cdot X + C
  \]

- **Move term right:**
  \[
  A \cdot Y
  \]
  COLCT
  \[
  A \cdot X = -(A \cdot Y) + B \cdot X + C
  \]

- **Move term left:**
  \[
  B \cdot X
  \]
  COLCT
  \[
  A \cdot X - B \cdot X = -(A \cdot Y) + C
  \]

- **Merge terms:**
  \[
  &1 \cdot &2 - &3 \cdot &2' 
  \]
  MATCH
  \[
  (A-B) \cdot X = -(A \cdot Y) + C
  \]

- **Isolate:**
  DROP 'X' ISOL
  \[
  X = (- (A \cdot Y) + C)/(A-B)
  \]

The final result is formally equivalent to the result obtained previously using RULES operations.

These examples illustrate the operation of EXPAN and COLCT compared with the use of RULES operations. Both approaches allow you to alter the form of an expression without changing its formal value. EXPAN and COLCT are “extensive” manipulation commands—they perform wholesale rearrangements potentially involving many terms in the same expression. Both have the shortcoming of trying to do many operations at once without providing you with any control of the precise form of their results. In many cases, you may be surprised or dissatisfied with the results because they don’t match some particular form that you desire. For example, the expansion of $x^3$ could return any of the formally equivalent expressions $xxx$, $x^2 x$, and $xx^2$. There is no
“correct” choice; EXPAN happens to return $x^2$:

$$'X^3' \text{ EXPAN } \equiv 'X*X^2'. $$

You can obtain the other choices by using RULES to commute the arguments of the * (switching $'X*X^2$ to $'X^2*X'$), or by using EXPAN again to obtain $'X*(X*X)'$.

The RULES approach, on the other hand, suffers from being too specific. That is, it allows you to rearrange expressions into a wide variety of equivalent forms, but you must execute one careful step at a time. The path of individual operations you need to follow to change an expression from one form to another requires some thought. RULES also suffers from the general EquationWriter problem of display speed--for large expressions, it takes a long time to show the result of each successive operation.

The best approach for general use of these three commands is to use COLCT and/or EXPAN one or more times on an expression to get it roughly into the form you want. Then use RULES to rearrange parts of the expression, until you obtain the desired final version. To speed up the use of RULES on complicated expressions, you can extract subexpressions using \texttt{ZSUBZ} in the subexpression menu, rearrange the subexpressions using RULES, then substitute the subexpressions back into the main expression using \texttt{EREPZ}.

In the next two sections we will describe the operations of COLCT and EXPAN. In most cases you will not need to follow their workings to anywhere near the detail we present. You may even want to skip these sections at a first reading. Typically, it is easier to use these commands and “see what you get” than to try to predict the outcomes exactly. Keep in mind that the net value of an expression is the same before and after you apply either of the commands; you use them only to rearrange the expression, either as a preliminary to ISOL, or to change the order of calculations, or just to recast it into a more familiar form.

\subsection*{17.1.1 COLCT}

The purpose of COLCT is to simplify an expression by combining sum and difference terms that differ only in their numerical coefficients. COLCT tries to reconstruct each term of an expression into the form $c*\text{subexpression}$, where $c$ is a real or complex number, and then combines all terms with the same \texttt{subexpression} by adding their coefficients $c$. Any terms that consist only of numbers are combined into a single term. In addition to this simple reconstruction, COLCT tries to improve the identification of like terms by applying some standardization to the terms:
Functions whose arguments are numbers are executed. In the expression 'X+5*SIN(30)', the arguments of SIN and * are numerical. COLCT therefore returns 'X+2.5' (in degrees mode).

Factors (arguments of *) in a term are put into a standard order, and combined into powers where appropriate:

'X*Y*X' COLCT \(\Rightarrow\) 'X^2*Y'.

Factors of similar quantities raised to powers are combined by adding their exponents:

'(X+Y)^(Z*(X+Y))' COLCT \(\Rightarrow\) '(X+Y)^(Z+T)'.

Even these rearrangements aren't enough to ensure that all terms which may appear suitable are actually combined by COLCT. For example, consider '2*(X+Y)+X'. You might expect COLCT to modify this expression to '3*X+2*Y'. However, COLCT leaves this expression unchanged, because COLCT does not distribute multiplication (in this case, it does not expand '2*(X+Y)' into '2*X+2*Y'). Without distribution, the terms '2*(X+Y)' and 'X' do not contain common non-numerical factors, so they are not combined.

The basic operation performed by COLCT is association--the reordering of the arguments and functions in multiple sums and differences, and the reordering of factors in multiple products (and quotients). The associative property of addition (and subtraction) means that the order of addition doesn't matter: \(a + (b + c)\) has the same value as \((a + b) + c\). Similarly, \((a-b)c\) has the same value as \(a(b-c)\); this is the associative property of multiplication. COLCT applies these rules systematically throughout an expression. An easy way to understand COLCT is to view an expression in its RPN form; for example, \((a + b) + (c + d + e)\) is

\[
a b + c d + e + +
\]

Here the summands (arguments of +) \(a, b, c, d,\) and \(e\) represent any subexpressions that do not consist of + (or −) and its two arguments, such as \(2x\) or \(\sin(3x + 4)\). COLCT rearranges this expression by

1. Moving all of the summands \(a, b, c, d,\) and \(e\) to the left, and the + operators to the right:

\[
a b c d e + + + +
\]

2. Sorting the summands into a standard order.
3. Combining consecutive summands that are the same except for a numerical coefficient by adding the coefficients.

This process is applied recursively to the individual summands, so that collection of terms takes place at several levels at once.

To illustrate the process, consider the expression \( Y + 2 \times (X + Y + X) + (X + Y + 3 \times X) \). In RPN form, this is

\[
Y \ (2 \ X \ Y \ + \ X \ + \ *) \ + \ X \ Y \ + \ (3 \ X \ *) \ + \ +
\]

Here we have inserted parentheses to help mark the sequences in which independent collection can take place. Now apply COLCT:

1. Move the summands to the left:
   \[
   Y \ (2 \ X \ Y \ X \ + \ + \ *) \ X \ Y \ (3 \ X \ *) \ + \ + \ + \ +
   \]

2. Sort the summands:
   \[
   (2 \ X \ X \ Y \ + \ + \ *) \ X \ (3 \ X \ *) \ Y \ Y \ + \ + \ + \ +
   \]

3. Combine consecutive common terms:
   \[
   (2 \ (2 \ X \ *) \ Y \ + \ *) \ (4 \ X \ *) \ (2 \ Y \ *) \ + \ +
   \]

Converting back to algebraic form, this is

\[
'2\times(2\times X + Y) + 4 \times X + 2 \times Y'
\]

Step 2, sorting the summands, may appear a bit mysterious. The sorting is necessary for the combination of like terms, but often the final order produced by the sorting does not correspond to any obvious rules. For example, \( 'A + B + Q + R' \) COLCT ⇨ \( 'A + Q + B + R' \), so the sorting is not a simple alphabetization. Actually, the ordering algorithm used by COLCT is designed for optimum speed, and depends on the idiosyncrasies of the HP 48 CPU and the way it stores bits of information in memory. The design of COLCT emphasizes speed rather than some form of standard ordering also because any choice of "standard" ordering would be arbitrary, and generally as likely to be "right" or "wrong" as any other, including the one actually used.

### 17.1.2 EXPAN

Although the usual effect of COLCT is to make an expression "smaller" by combining terms, and of EXPAN is to make it "bigger" by expanding products of sums, you should not consider them as inverses of each other. Whereas COLCT is based on the associative properties of addition and multiplication, EXPAN is derived from the distribution of products, quotients, and powers of sums. The simple distribution rules are
• **multiplication**: \( a(b + c) = ab + ac \).
• **division**: \( (b + c)/a = b/a + c/a \).
• **involution (powers)**: \( a^{b+c} = a^b a^c \).

Each of these rules has a straightforward representation in the actions of EXPAN:

\[
\begin{align*}
'A*(B+C)' & \text{ EXPAN } \Leftrightarrow 'A*B+A*C' \\
'(B+C)*A' & \text{ EXPAN } \Leftrightarrow 'B*A+C*A' \\
'(B+C)/A' & \text{ EXPAN } \Leftrightarrow 'B/A+C/A' \\
'A^(B+C)' & \text{ EXPAN } \Leftrightarrow 'A*B*A*C' \\
\end{align*}
\]

(You can also substitute \(-\) for \(+\) in the above examples.)

When both arguments of a product are sums, only the second sum is distributed:

\[
'(A+B)*(C+D)' \text{ EXPAN } \Leftrightarrow '(A+B)*C+(A+B)*D'
\]

There are two additional special cases of the distribution of powers:

• Expressions of the form \( a^n \) expand to \( a\cdot a^{n-1} \), where \( n \) is a positive integer real number. For example,

\[
'A^5' \text{ EXPAN } \Leftrightarrow 'A*A^4'.
\]

• Squares of sums are expanded from \((a+b)^2\) to \(a^2 + 2ab + b^2\):

\[
'(A+B)^2' \text{ EXPAN } \Leftrightarrow 'A^2+2*A*B+B^2' \\
'SQ(A+B)' \text{ EXPAN } \Leftrightarrow 'A^2+2*A*B+B^2'
\]

It is also possible to distribute the logarithm of a product into a sum or difference of logarithms:

\[
'LN(A*B)' \text{ EXPAN } \Leftrightarrow 'LN(A)+LN(B)' \\
'LN(A/B)' \text{ EXPAN } \Leftrightarrow 'LN(A)-LN(B)'
\]

EXPAN distributes the antilogs of sums and differences as follows:
Expression Manipulations

'EXP(A+B)' EXPAN ⇐ 'EXP(A)*EXP(B)'

'EXP(A-B)' EXPAN ⇐ 'EXP(A)/EXP(B)'

Similar expansions hold for the base 10 versions of these functions (LOG and ALOG).

These cases cover all of the potential rearrangements performed by EXPAN, if you generalize them by letting A, B, and C stand for any subexpressions. However, EXPAN does not necessarily make all possible expansions in an expression; specifically, EXPAN does not expand any subexpressions that are part of a distribution. For example,

'A*(B*(C+D)+E*(F+G))' EXPAN ⇐ 'A*(B*(C+D))+A*(E*(F+G)).'

To understand this example, it's useful to write the expression in Polish notation (section 2.1):

*(A , +( *(B,+(C,D)) , *(E,+(F,G)) ) )

EXPAN works into the expression, looking for products for which at least one argument is a sum, i.e. patterns of the form *(+(a,b),c) or *(a,+(b,c)). When it finds one in any subexpression, it distributes the multiplication, then does not attempt any further operations on the arguments of the sum. In the current example, the outermost subexpression is such a product, so the multiplication is distributed. The arguments of the sum, (B*(C+D)) and (E*(F+G)), are not expanded, even though they themselves are products of sums.

If you expand the example expression twice with EXPAN, the "inner" products are expanded. The result of the first expansion looks like this in Polish form:

+( *(A , *(B,+(C,D)) ) , *(A , *(E,+(F,G)) ) )

Now the outermost subexpression is a sum, which is not a candidate for expansion. Therefore, in the second use of EXPAN, each of the arguments of the outer sum is considered in parallel. Both arguments are products, but neither is a product of a sum, so the analysis branches again, into four subexpressions--the two arguments of each of the "outer" products. Of these, two--*(B,+(C,D)) and *(E,+(F,G))--are suitable for expansion, and are duly expanded, completing the operation in those branches. The other two branches--the two A's--are dead ends, so the expansion is complete, and the second EXPAN returns

'A*(B*C+B*D)+A*(E*F+E*G))'
17.2 The EquationWriter Subexpression Mode

The EquationWriter includes a powerful adjunct that we will call subexpression mode, from the capabilities it provides for operations on individual subexpressions (the Owner's Manual calls this the selection environment). These operations range from command-line editing to the application of various identity transformations. We introduced this mode in section 6.7.6, with a brief discussion of the associated subexpression menu; here we will elaborate on some of the topics, and focus on the operations collectively called RULES.

Subexpression mode is one of three modes in which you can operate the EquationWriter. You can always tell which of the three is active by considering the cursor and the menu:

- In entry mode, where you can extend the currently displayed expression with additional objects, the cursor is an open box at the right end of the expression:

\[
\sum_{N=0}^{I} \frac{X^I}{N+1} + \sqrt{\frac{X^3}{2+X}} + \cos(Z)^2
\]

The menu can be any ordinary command menu.

- In subexpression mode, the cursor is an inverse highlight of one object or subexpression:
The menu is either the main subexpression menu, as shown here, or one of the various RULES menus (section 17.2.3).

- In viewing mode, no menu appears:

\[
1 + 2 \cdot \sum_{N=0}^{\infty} \frac{X}{N+1} + \sqrt{\frac{X^3}{2+X}} + \cos(z)
\]

This mode lets you view an expression that is too large for the display. Pressing any arrow key scrolls the display window in the indicated direction over the expression. If you enter viewing mode from entry mode, the cursor disappears. If you enter from subexpression mode, the cursor remains, and is reactivated in place when you exit viewing mode.

You can activate viewing mode from either other mode by pressing [GRAPH]. Pressing that key again, or [ATTN], returns to the previous mode. To switch from entry mode to subexpression mode, press [J] (which suggests moving “back” into the expression). Initially, the rightmost object in the current expression is highlighted. To return to entry mode, press [EXIT]. In either of these two modes, [ENTER] exits from the Equation-Writer, with the modified expression entered into level 1. [ATTN] exits as well, but discards the Equation-Writer expression.

Note that you cannot enter subexpression mode from entry mode unless the active expression is complete—there are no functions without all of their arguments. Pressing [J] in such a situation flashes the Incomplete Subexpression warning and retains entry mode.

The subexpression menu contains the following entries, which we will discuss in the next sections:

- **RULES** activates a context-sensitive menu of transformations that may be applied to the selected subexpression.

- **EDIT** copies the selected subexpression into the command line for editing as text.
• \texttt{EXPR} extends the selection cursor to highlight the entire subexpression defined by the selected object.

• \texttt{SUB} copies the selected subexpression to the stack.

• \texttt{REPL} replaces the selected subexpression with a number or a symbolic object from level 1 of the stack.

• \texttt{EXIT} returns to the EquationWriter entry mode, with the open box cursor at the right end of the expression.

17.2.1 Navigating the Expression

The subexpression highlight cursor serves to select one object at a time in the displayed expression. For example, enter the following expression:

\[
\frac{A^2 + B^2 + C + F_0}{D + E^3}
\]

Now press \(<\) to activate subexpression mode:

The four cursor keys, plus their right-shifted (\(\uparrow\)) extensions, move the selection cursor from object to object. The involution function \(^\wedge\) is a special case, since it is not normally visible in the EquationWriter. But it does appear when you select it--with the \(F\) highlighted, press \(<\,\leftarrow\,\Delta\,\leftarrow\,\uparrow\):
Expression Manipulations

As you move the cursor around an expression, it highlights one object at a time. However, for the purposes of the various menu operations, the cursor actually selects the entire subexpression defined by the highlighted object and its arguments, if any. You can make the entire selection explicit by pressing EXPRE or +/- (the latter is always available, whereas EXPRE is not accessible when any of the RULES menus are active). For example, with the cursor still on the ^, pressing EXPRE highlights its subexpression:

The rightmost + is the top-level function in this expression, so moving the cursor there and pressing EXPRE highlights the entire expression:
17.2 Expression Manipulations

Notice that whenever you move the cursor, it always reverts to the single-object highlight; otherwise the constant size-changing of the cursor as you move it about an expression would be distracting and confusing. You can also shrink a subexpression highlight back to a single object by pressing \texttt{EXPR} or \texttt{+/-} again.

Although the highlight cursor motion is always generally in the direction indicated by the key arrows, it is sometimes difficult to predict just which object is next as you move the cursor. For example, with the cursor like this

\[ \frac{A^2 + B^2 + C + F}{D + E^3} \]

pressing \texttt{A} moves the cursor to the divide bar

\[ \frac{A^2 + B^2 + C + F}{D + E^3} \]

rather than to the 3 exponent, apparently because the 3 is not directly above the E. There is, of course, a complicated set of rules that determines just how the cursor will move. But it is not worth the trouble to list or learn the detailed rules. A little trial and error in ambiguous situations is faster than trying to analyze an expression yourself to save a few keystrokes.

17.2.2 Editing Subexpressions

It is useful to distinguish between two types of modification that you can apply to expressions as they are represented in the HP 48. The first, which we will call \textit{editing}, is
the alteration of the text form of an expression without regard to its original mathematical structure. The second, called *transformation*, is the application of various mathematical rules to subexpressions, preserving the structure of the overall expression and usually its formal value as well. Typically, you *edit* an expression when you have entered it incorrectly or when you are using it as a typing aid to create a near copy of the expression. You *transform* an expression when you are trying to solve it or re-express it in a different but equivalent form. In this section, we will discuss editing in subexpression mode; transformation is described in section 17.2.3.

**EDIT** copies the selected subexpression to the command line, where you can alter it freely using any of the normal facilities of the command line. There are two reasons for using the command line from the EquationWriter:

- You are already entering an expression with the EquationWriter, and you need to correct a previously entered portion.

- You wish to edit a complicated expression, where it may be easier to use the EquationWriter to pick out a portion for editing than to decipher the maze of parentheses that appears in the linear format display of the expression.

To illustrate the process, consider an expression entered and selected like this:

\[
\frac{A^2 + B^2 + C}{D+E^3} + F
\]

**EDIT** copies the subexpression defined by the + into the command line:
When you have finished editing the subexpression, [ENTER] restores the EquationWriter environment, with the new subexpression replacing the old. If you press [ATTN] when the command line is active, the original EquationWriter expression is restored. In the example, changing the C to C^3/2, then pressing [ENTER] yields this picture:

\[
\frac{A^2 + B^2}{2^3} + C^3 + D + E^3 + F
\]

There are a few restrictions on the contents of the command line that you can send back to the EquationWriter:

- Only one object can be entered.
- The object must be of a type suitable for embedding in an expression: a real or a complex number, a name, a unit object or an algebraic object.
- If the original selected subexpression was a name-only field, e.g. a variable of integration or differentiation, then the returned object must also be a name.

If the command line contains a syntax error, then the Invalid Syntax error is reported, and the command line remains active for your corrections. If one of the above rules is violated, then the calculator beeps but returns to the EquationWriter, with the original expression unchanged. However, the command line text is saved in the command stack, where you can retrieve it by pressing [EDIT] again, followed by [CMD].

The zSTK operation is available during a subexpression edit, for copying stack objects into the subexpression. If the EquationWriter was activated by [V] with a stack object, that object will not appear on the stack during the edit. Thus if you want to copy object_2 into object_1, enter object_2 into level 2, and object_1 into level 1. Then press [V] to start the EquationWriter, activate subexpression mode, select a subexpression, and press [EDIT]. At this point, zSTK will show object_2 in level 1, where you can use COPY to copy it into the command line. ENTER then returns the modified object to the EquationWriter environment.

17.2.3 RULES
The zRULES key in the subexpression menu activates a menu of manipulation operations. These operations, which we are collectively calling the RULES operations,
correspond to various mathematical rules for rearranging expressions (hence the name). The menu that you see when you press \texttt{RULES} varies according to the selected subexpression. At any time, the menu contains only those operations which may be applicable to the function that defines that subexpression. Some operations appear in all of the menus; others appear only for certain functions.

The selected function's arguments determine which of the available operations are usable in each individual case. For example, the \texttt{RULES} menu for * includes keys for moving factors, commuting, associating, distributing, merging, double negating, double inverting, and replacing the product of a logarithm with the logarithm of a power. In most cases, only a few of these can be applied. For \( A \times B \), the only options are commutation (to \( B \times A \)), double negation (to \( -(-A \times B) \)), and double inversion (to \( \text{INV(INV(A)/B)} \)).

We used \texttt{RULES} manipulations as part of an overall equation solving process in the example in section. That example illustrates the basic use of \texttt{RULES}:

1. Activate the EquationWriter subexpression mode with an expression or equation.
2. Move the cursor to select an object; if necessary, use \texttt{EXPR} or \texttt{+/=} to verify that you have selected the correct subexpression.
3. Press \texttt{RULES} to activate the operations menu for the selected object.
4. Press an operation key.
5. Repeat steps 2-4 until the expression is rearranged as desired.
6. Press \texttt{ENTER} to place the revised expression on the stack, or \texttt{EXIT} to go to EquationWriter entry mode.

As another example, consider rearranging the expression \( X^{(B+A)} + X^{(C+B)} \) to factor out the common factor \( X^B \). Starting with the expression in EquationWriter subexpression mode:

![Expression Manipulations](image)
First, commute the second exponent's sum, so that the B has the same position in both exponents. Move the cursor to highlight the +, by pressing ▼ once:

Now press RULES —— :

Now distribute the exponent. Press ▼ ▼ to highlight the ^, then RULES D—— :

Repeat the distribution for the $X^{B+A}$. Press ▼ eight times to highlight the ^, then RULES D—— :
Finally, press $\mathbf{\triangleright}$ four times to highlight the $+$, then $\boxed{\text{RULES $\pm$ ME}}$:

RULES operations share with EXPAN and COLCT the fundamental property of being identity operations, which never change the formal value of an expression. You can use them to rearrange an expression into a form more suitable for further calculations, with confidence you can't make mistakes that alter the symbolic value of the calculation that the expression represents. This differs significantly from the ordinary syntax checking that the command line editor performs.

The RULES operations are the ultimate "conservative" HP 48 symbolic manipulation system. They are primarily "single-step" operations that allow you to rearrange an expression into almost any form you want, making no changes that you don't explicitly specify. Of course, this means that a substantial rearrangement of an expression can require a long series of operations.

The expression manipulation facility called RULES here first appeared in the HP 28 under the name FORM. That implementation has two major problems:

- Subexpression selection is done with the linear format representation of algebraic objects, with all parentheses shown explicitly (such as with flag $-53$ set in the HP 48). With the line-wrapping necessitated by the 23-character display, this format makes it very difficult to identify subexpression structure.
• All transformations are strictly oriented to the calculator's representation of expressions as a subexpression hierarchy. That is, the HP28 "sees" the expression \( a + b + c + d + e + f \) as nested sums, i.e. \(((a + b) + c) + d + e + f\). Each sum is on a different level, and may contain one or more of the other sums as one of its arguments. We humans, on the other hand, tend to view the expression as a series of terms, each on the same level, more like \( a + b + c + d + e + f \). One of the most common pencil-and-paper operations is the simple reordering of the terms, such as moving the \( a \) to the right end: \( b + c + d + e + f + a \). This apparently simple rearrangement is a formidable task in HP28 FORM, involving many commutation and association operations.

These problems were addressed in the migration of FORM into HP 48 RULES. First, FORM's linear format representation is replaced in the HP48 by the EquationWriter subexpression mode. The presentation of an expression in a two-dimensional format with familiar graphical constructs makes it quite easy to pick out functions and their arguments. The \( \text{EXPR} \) key lets you confirm visually that the function you select does span the subexpression you want.

Second, the HP 48 adds new term movement operations, \( -T \) and \( T- \), which allow you easily to move summands and factors. For example, the rearrangement described above becomes a one-step operation. With the left-most + selected:

\[
\begin{align*}
\text{a+b+c+d+e+f} \\
\text{RULES EDIT EXPR SUB REPL EXIT}
\end{align*}
\]

\( \text{RULES} \ \text{[R]} \ \text{T-} \) moves the \( a \) all the way to the right:

\[
\begin{align*}
b+c+d+e+f+a \\
\text{DNEG UNY } ^*1 \ ^1 \ ^1 \ +1-1
\end{align*}
\]
This rearrangement is a very tedious exercise on the HP 28.

Of course, the (lack of) speed of the HP 48 Equation Writer remains as a deterrent to the use of RULES operations to replace pencil-and-paper calculations. Whether the latter is any faster than using RULES for a given problem is questionable, but at least when you do a problem by hand, you are kept busy constantly so that the time seems to pass more quickly.

17.2.3.1 Repeated Operations
HP 48 RULES adds one more feature that is not present in HP 28 FORM: automatic repeated operations. This means that in any situation where it makes sense to execute an operation more than once consecutively on the same object, you can use the right-shifted menu key as a shortcut. This applies to any of the following operations: £D-Z, -D, A-, -, M-, -, T-, and -. Pressing [®] followed by any of these keys executes the associated operation repeatedly on one object until no further change is possible. For example, you can distribute this entire product

\[ A \times (B+C+D+E+F) \]

at one stroke by pressing [®] RULES [®] D-

\[ A \cdot B + A \cdot C + A \cdot D + A \cdot E + A \cdot F \]

Similarly, you can merge the entire sum of products in one operation. From here
17.2 Expression Manipulations

A-B+A-C+A-D+A-E+A-F

pressing \texttt{RULES} \texttt{E-M} yields

A\cdot(B+C+D+E+F)

\textbf{17.2.3.2 Condensed RULES Notation}

Although the use of screen pictures is helpful in explaining the various RULES operations, it can also spread out a description so far that it is hard to see it all at once. Accordingly, in the following sections we will use a condensed notation to represent operations, where the “before” and “after” expressions are shown together on a single line, connected by the familiar $\equiv$ symbol. The selected object is indicated by a box around it, and the RULES operation is represented by its menu keybox. Thus the first step in the previous example would be shown as

$A \equiv (B+C+D+E+F) \implies D-2 \equiv A-B+A-C+A-D+A-E+A-F$

\textbf{17.2.3.3 Moving Terms}

As mentioned previously, we generally view expressions as a sequence of terms added (or subtracted) together. Because of the associative and commutative properties of addition, the order of terms can be changed without affecting the formal value of an expression. In manual calculations, this is one of the most common operations in the process of rearranging or solving. For example, in beginning algebra classes we are
taught to move all of the terms containing an unknown to one side of an equation and the remaining terms to the other, or to put the terms of a polynomial in order of ascending or descending power. To provide this same ability for symbolic objects, the HP 48 RULES system includes two operations, \texttt{\textasciitilde T (term left)} and \texttt{T\rightarrow (term right)}, for moving terms left and right in an expression. Furthermore, these operations incorporate factors---arguments of \texttt{\textasteriskcentered} and \texttt{/\textendash} in their generalized definition of term, so that you can reorder products and quotients as well.

\texttt{\textasciitilde T} moves a term one position to the left:

\[ A + B \oplus C + D \quad \texttt{\textasciitilde T} \quad A \oplus C + B + D \]

There are several things to note in this example:

- The selected object for the operation is the \texttt{\textasciitilde} to the left of the term that is moved, not the object \texttt{C} that is actually moved. This is because the term operations are associated with the functions \texttt{\textasciitilde}, \texttt{-}, \texttt{*}, \texttt{/} and \texttt{=} \texttt{\textasciitilde}, rather than with than their arguments.

- The term moved left is to the right of the originally selected function. When a function is selected for a term operation, it defines the two terms on either side as the “candidates” for the operation. With that limited focus, it is only the right term that can be moved left, and the left term that can be moved right. For single movements, \texttt{\textasciitilde T} and \texttt{T\rightarrow} could have been combined in a single “local commute” operation. The two separate operations are provided to aid in repeated movements of one term in one direction.

- After the operation, the selection cursor has moved to the next \texttt{\textasciitilde} in the direction of the term motion, staying to the left of the moved object \texttt{C}. Thus to move that object further in the same direction, just push \texttt{\textasciitilde T} again:

\[ A \oplus C + B + D \quad \texttt{\textasciitilde T} \quad C \oplus A + B + D \]

After the second movement, there are no more \texttt{\textasciitilde}’s to the left, so the selected cursor ends up on the \texttt{C}.

In general, you can move a term continuously in one direction by pressing \texttt{\textasciitilde T} or \texttt{T\rightarrow} repeatedly. To move a term all the way in either direction, use \texttt{\textasciitilde T} or \texttt{\textasciitilde T\rightarrow }, which yield the same results as using the unshifted versions repeatedly until there is no further motion.

Term operations apply to factors as well:
When sums and products are intermixed, the extent of a particular term and the range of its potential motion depends on the order of the functions. This is best illustrated by examples:

\[ A - B + C + D \cdot E + F + G \cdot H \cdot I \rightarrow T \rightarrow A \cdot B + C + D \cdot E + F + G \cdot H \cdot I \]

\[ A - B + C + D \cdot E + F + G \cdot H \cdot I \rightarrow T \rightarrow A \cdot B + C + D \cdot E + F \]

In effect, sequences of `+`'s (or `-`'s) and `*`'s (or `/`'s) form a kind of term hierarchy; a particular term can be moved around within its own sequence, but not beyond.

In most circumstances, `=` is treated the same as `+` or `-` for term movements, except for a sign change as a term moves across the equals sign:

\[ A + B + C = D \rightarrow T \rightarrow A \cdot B = -C + D \]

However, an exception is made for the case where one side of an equation is a product or quotient. Instead of moving the entire product or quotient across the equals sign, `T-` or `-T` moves only the factor nearest the `=`, dividing or multiplying the expression on the opposite side:

\[ A \cdot B = -C + D \rightarrow T \rightarrow A = INV(B) \cdot (-C + D) \]

\[ \frac{A}{B} = C \cdot D \rightarrow T \rightarrow A = B \cdot (C \cdot D) \]

This exception is provided for the purpose of solving an equation for a variable, where you have isolated the variable on one side of the equation save for a multiplicative coefficient.

17.2.3.4 Commutation

*Commutation* is the exchange of the arguments of a two-argument function. The commutative laws of arithmetic can be summarized as

\[ a + b = b + a \quad \text{Addition} \]
17.2

Expression Manipulations

\[ a - b = -b + a \]  \hspace{1cm} \text{Subtraction}

\[ a \cdot b = b \cdot a \]  \hspace{1cm} \text{Multiplication}

\[ \frac{a}{b} = \frac{1}{b} \cdot a \]  \hspace{1cm} \text{Division}

A typical use of commutation (\(\equiv\)) is to reorder the terms of an expression so that terms with a common factor can be grouped together, as a preliminary to factoring out the common factor. For example, reordering \(ax + by + cx\) to \(by + ax + cx\) is done by commuting the arguments of the first +.

### 17.2.3.5 Association

Association is a change in the precedence of calculation, the order in which the operations are carried out. A common form of association is the conversion of \(a + (b + c)\) into \((a + b) + c\). The fact that this represents a change in the order of calculation is easily apparent when you write the two expressions in RPN form, in which calculation proceeds from left to right:

- \(a + (b + c)\) is \(a \ b \ c + +\)
- \((a + b) + c\) is \(a \ b + c +\)

There are two functions involved in any association (the two +'s in each example); for RULES, you must select the one that defines the entire subexpression that is associated. In \(a + (b + c)\), the first + should be selected; in \((a + b) + c\), the second. Furthermore, you need to specify a “direction” for the association. In an expression like \((a + b) + (c + d)\), the middle + can be selected for association in combination with either of the other two +’s. \(\equiv\text{-A}\equiv\) (associate left) works when the selected operator is to the left of the second operator, moving the parentheses to the left. For example,

\[ (A+B) \text{ [+] (C+D) } \equiv\text{-A}\equiv \ L ( (A+B) + C ) \text{ [+] D.} \]

(Here we have set flag -53 to show all of the implied parentheses.) Similarly, \(\equiv\text{A}\equiv\) works when the selected operator is on the right. The choice of “right” and “left” in the operations’ names is rather arbitrary--equally good reasons could be offered for reversing the names. It’s often easier to try one of the two choices and see if you get what you want, than to remember which is which. If you get the wrong effect, then use the opposite operation twice.
After execution of $A\rightarrow$ or $\leftarrow A$, the selected object is that function that defines the subexpression that contains all of the same objects as the original selected subexpression.

Additional operations provide for multiple associations in one step.

- $(())$ parenthesizes the sum-of-nearest-neighbors, or product-of-nearest-neighbors, ignoring intervening parentheses:

\[
A + B \left[ (+) \right] C + D \xrightarrow{((())\\rightarrow)} A \left[ (+) \right] (B + C) + D
\]

This operation has no apparent effect if the selected function is the first (or only) in the sequence, since these parentheses are already present, but hidden.

- ($\downarrow$) moves the parenthesis on the left of the selected subexpression to include the next term to the left (which may result in a matched pair of parentheses disappearing):

\[
A + B + (C \left[ (+) \right] D) + E \xrightarrow{\downarrow\downarrow} A + (B \left[ (+) \right] C + D) + E
\]

\[
A + (B \left[ (+) \right] C) + D \xrightarrow{\downarrow\downarrow} A + B + C + D
\]

- $\leftarrow$) moves the parenthesis on the right of the selected subexpression to include the next term to the right:

\[
A + (B \left[ (+) \right] C) + D \xrightarrow{\leftarrow\leftarrow} A + (B + C \left[ (+) \right] D)
\]

17.2.3.6 Distribution

Distribution operations allow you to perform EXPAN-like expansions on individual functions and their associated subexpressions. Like association, distribution involves two functions, and you must select the one that defines the subexpression containing both. For example, to distribute the multiplication in the expression $A \cdot (B + C)$, you select the $\cdot$, then press $\rightarrow D$. This returns $A \cdot B + A \cdot C$, in which the $+$ is highlighted because it is the defining object for the new subexpression.

Two distribution operations are necessary because of the ambiguity of expressions like $(a + b) \cdot (c + d)$. $D\rightarrow$ distributes the sum on the right:

\[
(A + B) \left[ (+) \right] (C + D) \xrightarrow{\rightarrow D} (A + B) \cdot (C + D) \xrightarrow{\rightarrow D} (A + B) \cdot C \left[ (+) \right] (A + B) \cdot D.
\]

$\leftarrow D$ distributes the sum on the left:
\[(A+B) \times (C+D) \equiv D \Rightarrow A \cdot (C+D) + B \cdot (C+D).\]

For logarithms and antilogarithms, which are functions of one argument, there is no ambiguity, and only D→ is allowed. Thus,

\[\ln (A\cdot B) \equiv D \Rightarrow \ln(A) \oplus \ln(B)\]

17.2.3.7 Merging

Merging is the inverse of distribution. That is, where distribution expands \(a \cdot (b + c)\) into \(a \cdot b + a \cdot c\), merging reverses the process, factoring \(a \cdot b + a \cdot c\) into \(a \cdot (b + c)\). Two forms of merge are necessary to handle ambiguous cases:

\[
A \cdot B \oplus A \cdot B \equiv M \Rightarrow (A + A) \cdot B,
\]

and

\[
A \cdot B \oplus A \cdot B \equiv M \Rightarrow A \cdot (B + B).
\]

Note that ←M and D→ are inverses of each other, as are M→ and ←D. ←M also handles the same logarithm and antilogarithm cases (in the opposite sense) as D→; for example

\[\exp(A) \oplus \exp(B) \equiv -M \Rightarrow \exp(A + B).\]

17.2.3.8 Prefix Operations

This group of operations is organized around the “prefix operators” ← and INV, which also happen to be their own inverses: \(a = -(−a)\) and \(a = INV(INV(a))\). The basic operation is distribute-prefix-operator, ←( ), which “pushes” one of these operators “into” its parentheses by altering the argument. For example,

\[\neg (A+B) \equiv \neg (\neg a) \Rightarrow \neg a \neg B\]

The inverse of ←( ) depends on the operator that is distributed.

- For negation, the effect of ←( ) is reversed by ←( ). The latter is called double negate and distribute, since it is equivalent to double negation (see below) followed by a distribution of a ← prefix operator. Examples:

\[\neg (A \cdot B) \equiv \neg \neg (\neg a) \Rightarrow \neg a \neg B.\]

\[\neg A \neg B \equiv \neg (\neg (\neg a)) \Rightarrow \neg (A \cdot B).\]
17.2 Expression Manipulations

- **Inverse of power or inverse of inverse-product:**

\[
\text{INV} \left( A^B \right) = A^{-B}
\]

\[
\text{INV} \left( \frac{\text{INV}(A)}{B} \right) = A^{-B}B.
\]

To return either of these subexpressions to its original form, you must use \(1/( -()\) , which inverts a subexpression by inverting its arguments. Another example of \(1/( -()\) is

\[
\text{EXP} (A) = \text{INV} \text{EXP}(-A).
\]

- **Double inverse:**

\[
\text{INV} \text{INV}(A) = A.
\]

The reverse operation is \(\text{DINV}\) (double-inversion) which takes any subexpression \(A\) and changes it into \(\text{INV}\text{INV}(A)\).

- **Double negative:**

\[
-\ A = \ A.
\]

The reverse operation is \(\text{DNEG}\) (double-negation), which takes any subexpression \(A\) and changes it into \(-(-A)\).

- **Complex Operators.** Distribution of negation and multiplicative inverse is extended to the common complex operations of conjugate (CONJ), real part (RE), and imaginary part (IM). Using \(-()\), you can “commute” the operator through certain other real-linear functions: CONJ with +, -, *, /, NEG, INV, and SQ. and RE and IM with +, -, *, and NEG.

\[
\begin{align*}
\text{CONJ} (\text{SQ}(A)) & \rightarrow \text{SQ} \text{CONJ}(A) \\
\text{RE} (A+B) & \rightarrow \text{RE}(A) + \text{RE}(B) \\
\text{IM} (A-B) & \rightarrow \text{RE}(A)\text{IM}(B) \pm \text{IM}(A)\text{RE}(B)
\end{align*}
\]

17.2.3.9 Unit Identities

The four simple identity operations \(*1, /1, ^1, \text{and} +1-1\), can be used with any subexpression. They are used as preliminaries for merging, in cases where some symmetry is lacking that prevents the merge from working. For example, if you want to factor \(A\cdot B + A\) into \(A\cdot (B + 1)\), you can’t use \(\text{M}\) or \(-\text{M}\) because the two arguments of the + are
not both products. You can achieve the factoring like this:

\[ A \cdot B^+ [A] = A \cdot B^+ [A] \]

\[ A \cdot B^+ [A] = A \cdot (B+1). \]

Another example:

\[ A^B \cdot A^+ = A^B \cdot A^+ \]

\[ A^B \cdot A^+ = A^B \cdot A^+ \]

/1 is the division analog of *1. Note that you can use *1 and /1 effectively as inverses of each other:

\[ A^1 = A \]

\[ A^1 = A \]

The last in this class of operations is +1-1, which you can use to split off terms as follows:

\[ 3 \cdot A = (3+1 \cdot 1) \cdot A \]

\[ (3+1 \cdot 1) \cdot A = (2+1) \cdot A \]

\[ 2 \cdot A+1 \cdot A = 2 \cdot A+1 \cdot A \]

17.2.3.10 Adding Fractions

The AF operation allows you to combine a sum or difference of two subexpressions, one or both of which are ratios, into a single numerator over a common denominator. The most general form of this operation is

\[ \frac{A}{B} + \frac{C}{D} = \frac{A \cdot D + B \cdot C}{B \cdot D} \]

The + in this example can be replaced by −. Two additional points:

- AF only requires one of the two original subexpressions to be a ratio— you can combine \(A + B/C\) into \((A \cdot C + B)/C\), for example.
• If the two subexpressions have the same denominator, use $M\rightarrow$ rather than $AF$, so that the final denominator is the same as the original.

17.2.3.11 Logarithms
$L^*$ and $L(\cdot)$ are a pair of operations based on the equivalence $\ln a^b = (\ln a)\cdot b$. $L^*$ transforms the log of a power $\ln(A^B)$ into the product $\ln(A)\cdot B$; $L(\cdot)$ reverses the transformation. Either works with natural logs ($\ln$) or common logs ($\log$). $L(\cdot)$ expects the log to be the first argument of the product; if you have the form $B\cdot \ln(A)$, you will have to commute the arguments with $\leftrightarrow$ before applying $L(\cdot)$.

17.2.3.12 Exponentials
$E^*$ and $E(\cdot)$ are based on the equivalences $e^{ab} = (e^a)^b$ and $e^{a/b} = (e^a)^{1/b}$. $E^*$ converts left-to-right in these equations, changing $\exp(A-B)$ into $\exp(A)^\cdot B$, for example. $E(\cdot)$ is the right-to-left operation. Either works with division as well as multiplication, and with ALOG instead of EXP.

17.2.3.13 Definition Expansions
The RULES menus for the trigonometric and hyperbolic functions include the $\rightarrow$DEF operation, which replaces these functions with their definitions in terms of EXP and LN. The particular forms of the expansions are chosen from various possibilities to exhibit the same branch-cut behavior as the corresponding HP 48 numeric functions. Table 17.1 shows an example of each operation, where the results are obtained in radians mode, and $X$ represents any subexpression.

The menu for EXP contains the $\rightarrow$TRG operation, which translates an exponential function into a sum of trigonometric functions:

$$\exp(X) \rightarrow \text{TRG} = \cos \left( \frac{X}{i} \right) + \sin \left( \frac{X}{i} \right) \cdot i$$

(The ALOG menu contains the same entries as the EXP menu, but $\rightarrow$TRG does not apply to ALOG.)

17.2.3.14 Addition Angle Formulae
The addition angle formulae for trigonometric and hyperbolic functions are provided as transformations by the TRG* operation (named by analogy with the multiply-argument operation for exponential, EXP^*). The specific transformations are listed in Table 17.2.
Table 17.1. ~DEF Expansions

<table>
<thead>
<tr>
<th>Function</th>
<th>~DEF Expansion</th>
</tr>
</thead>
</table>
| SIN(X)   | \[
\frac{\exp(X+i) - \exp(-X+i)}{2i}\]
| COS(X)   | \[
\frac{\exp(X+i) + \exp(-X+i)}{2}\]
| TAN(X)   | \[
\frac{\exp(X+2i) - 1}{\exp(X+2i) + 1}i\]
| SINH(X)  | \[-(i\sin(X))\]
| COSH(X)  | \[\cos(X)\]
| TANH(X)  | \[-i\tan(X)\]
| ASIN(X)  | \[-i\ln\left(\sqrt{1-X^2} + iX\right)\]
| ACOS(X)  | \[\frac{\pi}{2} + i\ln\left(\sqrt{1-X^2} + iX\right)\]
| ATAN(X)  | \[-i\ln\left(\frac{1+iX}{\sqrt{1+X^2}}\right)\]
| ASINH(X) | \[-\ln\left(\sqrt{1+X^2} - X\right)\]
| ACOSH(X) | \[\sqrt{-\left(\frac{\pi}{2} + i\ln\left(\sqrt{1-X^2} + iX\right)\right)^2}\]
| ATANH(X) | \[-\ln\left(\frac{1-X}{\sqrt{1-X^2}}\right)\]

Table 17.2. TRG* Transformations

<table>
<thead>
<tr>
<th>Function</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN(X+Y)</td>
<td>SIN(X)\cdot\cos(Y) + COS(X)\cdot\sin(Y)</td>
</tr>
<tr>
<td>COS(X+Y)</td>
<td>COS(X)\cdot\cos(Y) - \sin(X)\cdot\sin(Y)</td>
</tr>
<tr>
<td>TAN(X+Y)</td>
<td>[\frac{(\tan(X) + \tan(Y))}{1 - \tan(X)\cdot\tan(Y)}]</td>
</tr>
<tr>
<td>SINH(X+Y)</td>
<td>SINH(X)\cdot\cosh(Y) + COSH(X)\cdot\sinh(Y)</td>
</tr>
<tr>
<td>COSH(X+Y)</td>
<td>COSH(X)\cdot\cosh(Y) + \sinh(X)\cdot\sinh(Y)</td>
</tr>
<tr>
<td>TANH(X+Y)</td>
<td>[\frac{(\tanh(X) + \tanh(Y))}{1 + \tanh(X)\cdot\tanh(Y)}]</td>
</tr>
</tbody>
</table>
17.2.3.15 Collecting Terms
The final entry in all of the RULES menus is COLCT. This operation is useful in subexpression mode for simplifying by combining like terms. For example, you can transform \((1+2)\cdot X\) into \(3\cdot X\) by applying COLCT to the +.

In HP 48 versions through E, the COLCT operation in the RULES menu is a restricted "one-pass" version of the COLCT command. It does collect like terms, but it does not execute functions of numerical arguments other than the selected function itself. That is, \(2\cdot X + 3\cdot X\) collects only to \(2+3\) \[\cdot X\]; to simplify further to \(5\cdot X\), you must use COLCT again on the \(2+3\). Executed as a command outside of the EquationWriter environment, COLCT reduces \('2*X+3*X'\) to \('5*X'\) in a single execution.

17.3 Pattern Matching and Substitution
Ordinary algebraic object evaluation permits the substitution of expressions for names. That is, if \(X\) is defined as \('Y+1'\), then \('2*X'\) EVAL returns \('2*(Y+1)'\). However, this does not provide for the reverse substitution--replacing instances of \(Y+1\) with \(X\). More generally, there is an endless number of useful transformation rules that are not explicitly provided via EXPAN, COLCT, RULES, etc. To cover such cases, the commands \(\text{\texttt{\textasciitilde\texttt{MATCH}}}\) and \(\text{\texttt{\textasciitilde\texttt{MATCH}}}\) let you create and execute your own transformation rules. A simple example is the reverse substitution just mentioned:

\[
\text{\texttt{\textasciitilde\texttt{MATCH}}} '2*(Y+1)' \{ 'Y+1' 'X' \} '2*X' 1
\]

The 1 returned to level 1 is a true flag (section 9.3), which indicates that a successful match was made.

\(\text{\texttt{\textasciitilde\texttt{MATCH}}}\) and \(\text{\texttt{\textasciitilde\texttt{MATCH}}}\) are similar in operation, so we will refer to them collectively as MATCH except when the distinction is important. Both commands take as arguments an expression to rewrite, a pattern expression, and a replacement expression. The result is the original expression with the replacement expression substituted for every instance of the pattern expression. The pattern and the replacement are entered together in a list:

\[
\text{\texttt{\textasciitilde\texttt{MATCH}}} 'old' \{ 'pattern' 'replacement' \} 'new' \text{\texttt{\textasciitilde\texttt{flag}}}.
\]

The flag indicates whether any substitution was made. The flag is helpful to indicate when an iteration or recursion with MATCH is making no further changes in an expression.

MATCH allows the pattern and the replacement result expressions to contain "wild card" names which will match any subexpression. Any local or global name that starts
with the "&" character is interpreted as a wild card. MATCH replaces the wild card names in the replacement expression by the subexpressions each name matches in the original expression. If the same wild card variable is used more than once in the pattern expression, then it must match identical subexpressions. If a wild card occurs in the replacement expression without occurring in the pattern expression, the Undefined Name error results.

This powerful extension allows the straightforward application of rewrite rules. For example, the addition law for cosines can be applied to any expression by means of the following program:

```
COSSUM  Cosine of a Sum  E417

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 2</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'expression_1'</td>
<td>'expression_2'</td>
<td>flag</td>
</tr>
</tbody>
</table>

<< { 'COS(&A + &B)' |
     'COS(&A)*COS(&B) - SIN(&A)*SIN(&B))' |
   } |
   tMATCH |
   >>

Pattern. |
Replacement. |
```

COSSUM can work with simple expressions:

\[ \cos(X+Y) \] COSSUM \(\Rightarrow\) \[ \cos(X)\cos(Y) - \sin(X)\sin(Y) \] 1,

or with more complicated expressions:

\[ \ln(\cos(X^2 + 1/Y)) \] COSSUM

\(\Rightarrow\) \[ \ln(\cos(X^2)\cos(1/Y) - \sin(X^2)\sin(1/Y)) \] 1.

You can just as easily define the inverse transformation TOCOSSUM:

\[ \ln(\cos(X^2)\cos(1/Y) - \sin(X^2)\sin(1/Y)) \] TOCOSSUM

\(\Rightarrow\) \[ \ln(\cos(X^2 + 1/Y)) \] 1.
The pattern and replacement expressions are combined in a list in order to provide for an optional third argument, a conditional test expression. That is, the most general level 1 argument has the form

\[
\{ \text{pattern'} \text{'} replacement' test' } \).
\]

When MATCH is executed, the replacement is only substituted for the pattern if test evaluates to a true flag. Most commonly, a test is used to restrict the domain of a transformation. For example, simplification of the square root of a square is only valid when the argument of the square is positive:

\[
'\sqrt{3^2} + \sqrt{(-3)^2}' \{ '\sqrt{&1^2}' &1 '&1\geq0' \}
\]

\[
\text{IMATCH} \not\equiv '3 + \sqrt{(-3)^2}' 1
\]

The last refinement in the use of MATCH is the choice of tMATCH or tMATCH, which determines whether a pattern search is made upwards or downwards through a subexpression hierarchy (section 3.5.2.1). The distinction is illustrated by the following examples:

\[
'A+B+C' \{ '&1+&2' D \} t\text{MATCH} \not\equiv 'D' 1
\]

\[
'A+B+C' \{ '&1+&2' D \} t\text{MATCH} \equiv 'D+C' 1
\]

tMATCH looks for a pattern match starting at the highest level of the subexpression. In terms of the equivalent RPN sequence (section 2.1), this means that tMATCH works from right to left through the object sequence. In this case, the second + is the highest level function, which you can verify by setting flag -53 to show the original expression as '(A+B)+C', or by executing OBJ-:
The pattern expression \( &1+&2 \) matches any sum, so the entire expression is replaced by \( 'D' \).

\( \star\text{MATCH} \) works “from the inside out,” working from the lowest level subexpressions upwards (i.e., from left to right through the RPN object sequence). For this example, the subexpression \( A+B \) matches the pattern, so it is replaced by \( D \). However, the subexpression defined by the second + in the original expression, which is now changed to \( 'A+D' \), is not considered for replacement. This rule is followed by both \( \star\text{MATCH} \) and \( \star\text{MATCH}: \) A subexpression that has been matched once (and hence replaced), is not considered as a candidate for any further matches. Consider the following examples:

\[
\begin{align*}
'(A+B)+(C+D)' & \quad \{ 'A+B' 'C' 2 + \} \quad \star\text{MATCH} \quad \rightarrow \quad 'D' \\
'(A+B)+(C+D)' & \quad \{ 'A+B' 'C' 2 + \} \quad \star\text{MATCH} \quad \rightarrow \quad '(E+F)+(E+F)' 
\end{align*}
\]

Again, \( \star\text{MATCH} \) replaces the entire expression, because it matches the pattern at the top level (the middle +). However, it does not attempt to replace the new expression, even though it does match the pattern. The reason for the rule is quite apparent in this example: the replacement process would continue forever otherwise, as each successive replacement matches the pattern. Similarly, \( \star\text{MATCH} \) replaces the \( A+B \) and the \( C+D \), but not the expression defined by the middle +, because that sum’s arguments have already been replaced.

### 17.4 Simplifying Polynomials

Because of the generality of the HP 48 manipulation commands, they can be difficult to use for rearranging particular types of expressions into standard forms. A very common case is ordinary polynomials, which are sums of powers of a variable

\[
\sum_{n=0}^{N} a_n x^n.
\]

To expand the expression \( '(X+A)^4' \), for example, into a sum of powers of \( X \), you need to execute \( \text{EXPAN} \) and \( \text{COLCT} \) several times. It is quite difficult to predict how many executions of the commands are needed, and in what order they should be applied. Furthermore, to rearrange the result terms in order of ascending or descending powers requires the use of \( \text{RULES} \). You can do better by taking advantage of the fact that a Taylor’s series is a polynomial:
To use TAYLR, you must specify the name of the polynomial variable (level 2), and the order of the polynomial. If you don't know the order in advance, you can try any number that you are sure is larger than the actual order. You obtain the fastest execution if you specify a number equal to the polynomial order. This method is preferable to the blind use of EXPAN and COLCT because it is faster and more predictable.

The built-in commands are awkward to use in many cases because of their generality. You can do considerably better by making assumptions about the structure of certain expressions and taking advantage of the HP 48's ability to dissect objects. For polynomials, the assumption is that an expression is a sum of arbitrary coefficients times powers of a single variable. It can therefore be represented as a list of coefficients, with the corresponding powers indicated by the position of each coefficient in the list. Whether the lists are in ascending or descending order of powers is a matter of taste; we will use descending order in the programs presented below. For lack of a better term, we will use the term polynomialize to refer to the conversion of any expression into this standard form, either as an expression including the variable or as a list of coefficients. For example, \((X+A)^4\) polynomialized as a list becomes

\[
\{1 '4*A^3' '6*A^2' '4*A' 1\},
\]

where the \(X\) is implied, or stored as a separate object. Using the list representation, common mathematical operations can be represented by programs that use and preserve the list structure. In this section, we will describe a series of programs designed for operations with polynomials. The programs are listed all together at the end (section 17.4.1).

The first program, PADD, performs polynomial addition by adding the corresponding terms in two coefficient lists:

\[
\{A2 A1 A0\} \{B1 B0\} \text{PADD} \rightarrow \{A2 'B1+A1' 'B0+A0'\}.
\]

The programs for polynomial negation (PNEG), subtraction (PSUB), multiplication (PMULT), division (PDIVD), and involution (PPWR) use methods similar to those of PADD. PSUB could be written as a variation of PADD, but it is simpler to combine PNEG and PADD.

\[
\{A2 A1 A0\} \{B1 B0\} \text{PSUB} \rightarrow \{A2 '-B1+A1' '-B0+A0'\}.
\]
Polynomial multiplication is achieved by multiplying each term in one list by each term in the other, and adding up the corresponding products.

\[
\{A_2 \ A_1 \ A_0\} \ \{B_1 \ B_0\} \ \text{PMUL} \quad \Rightarrow \quad \{\ 'A_2*B_1' \ 'A_2*B_0+A_1*B_1' \ 'A_1*B_0+A_0*B_1' \ 'A_0*B_0' \ \}.
\]

Polynomial division is more difficult than the preceding operations, because it does not in general return a polynomial. The division program PDIV returns three lists, representing the remainderless quotient, the numerator of the remainder, and the denominator, respectively.

\[
\{1 \ 5 \ 7\} \ \{1 \ 2\} \ \text{PDIVD} \quad \Rightarrow \quad \{1 \ 3\} \ \{1\} \ \{1 \ 2\}
\]

represents the division of \(x^2 + 5x + 7\) by \(x + 2\), which simplifies to \(x + 3\) with a remainder of \(1/(x + 2)\).

The involution program PPWR considers only positive integer powers, so that the result is an ordinary polynomial. In this case, involution is equivalent to repeated multiplication of the base polynomial by itself. However, programming the involution that way is rather inefficient, since it involves computing the same products many times. On the other hand, using a full-blown multinomial expansion is an involved process requiring a substantial program just to compute the expansion coefficients. The approach used in PPWR is an intermediate one, in which any polynomial above first order is treated as a binomial:

\[
P_N(x) = a_Nx^N + P_{N-1}(x)
\]

An integer power of the polynomial can be computed from the binomial expansion:

\[
P_N^i(x) = \sum_{n=0}^{i} \binom{i}{n} (a_Nx^N)^n P_{N-1}^{i-n}(x)
\]

The powers of the lower-order polynomial \(P_{N-1}\) are computed recursively by the same method. This method works for any polynomial raised to any power, but it does require a lot of free memory for large polynomials and powers.

For example, using PPWR to expand \((x+2)^5\):

\[
\{1 \ 2\} \ 5 \ \text{PPWR} \quad \Rightarrow \quad \{1 \ 10 \ 40 \ 80 \ 80 \ 32\}
\]
The programs illustrated so far become even more useful when combined with other programs that convert between the list and symbolic object representations of a polynomial. The conversion from list to symbolic object is quite straightforward, as you can see in the program PEXPR. The level 1 argument for PEXPR can be any object allowed in an algebraic object:

{1 2 3} 'X' PEXPR \rightarrow 'X^2+2*X+3'

{1 2 3} 10 PEXPR \rightarrow 123

The conversion from an expression to a list is more difficult. One approach might be to use \texttt{tMATCH} or \texttt{tMATCH} (section 17.3) to replace the operators \(+, - , *, / \text{ and } ^\) in an expression with user-defined function versions of \texttt{PADD}, \texttt{PSUB}, etc., and convert names, numbers, and non-polynomial functions to zeroth-or first-order coefficient lists. Evaluating the result would then return the coefficient list for the whole expression. However, that method is difficult to adapt for functions other than those for which explicit polynomial versions are provided. Instead, the program \texttt{PLIST} takes an expression apart one level at a time using \texttt{OBJ-}. Applying this command to an algebraic object returns the “top-level” object, its arguments if any, and a count of those arguments:

\[ f(arg_1, arg_2, \ldots, arg_n) \rightarrow arg_1 \quad arg_2 \quad \ldots \quad arg_n \quad n \quad f \]

\texttt{PLIST} operates on a symbolic object, or any object that can appear in an algebraic object, plus the name of the designated polynomial variable. Its action depends on the type of the level 2 argument:

- For a real or complex number, or a unit object, or any name other than the polynomial variable name, \texttt{PLIST} returns the object in a one-element list, representing a zero-order polynomial.
- If the argument is a name that matches the polynomial variable, \texttt{PLIST} returns the list \{ 1 0 \}.
- If the argument is an algebraic object, it is dissected with \texttt{OBJ-}. If the top-level object is not one of the polynomial operators, then \texttt{PLIST} just returns the original object in a one-element list. For the two-argument operators, \texttt{PLIST} applies itself recursively to each of the arguments, then executes the appropriate polynomial list program to combine the arguments.
- Division is included as one of the special polynomial operators even though it does not return a polynomial in all cases, primarily to permit simplification of expressions where a denominator is actually a factor of its numerator, and so may be canceled out. Any remainder is converted back to a fraction by \texttt{PLIST}, and treated as a zero-
order term.

- PLIST calls LPWR for involution, rather than PPWR, since the second argument of ^ may be a general expression rather than just a real number. LPWR is an extension of PPWR that does additional argument screening and then calls PPWR if appropriate.

- If PLIST’s first argument is an equation, it still returns only one coefficient list, treating the = as a subtraction.

As an example of using PLIST, polynomialize \((x + 1)^3 - 4(x - 1) + 2\) into a list:

\[
(X+1)^3-4*(X-1)+2 \ 'X' \ PLIST \rightarrow \{1 \ 3 \ -1 \ 7\}.
\]

The goal set out at the beginning of this section is to devise a fast and automatic means of simplifying polynomials. The final step towards this goal is the program POLY, which polynomializes an expression into a new expression, “hiding” the intermediate list stages:

\[
(X+1)^3-4*(X-1)+2 \ 'X' \ POLY \rightarrow \ 'X^3+3*X^2-X+7'.
\]

or

\[
((3*X^2-3)/(X+1))^-2-(X+1)^2 \ 'X' \ POLY \rightarrow \ '8*X^2-20*X+8'
\]

Such reductions are essentially intractable using EXPAN, COLCT, and RULES. You can achieve similar results in some cases using TAYLR, but POLY is faster and does not evaluate any of the names in the argument expression.

POLY takes an arbitrary expression and a name as arguments, and returns a new algebraic object expressed as a polynomial in the named variable. POLY invokes PLIST to sort out the polynomial coefficients, then PEXPR to convert the coefficient list back into an expression.

When applied to a polynomial containing names other than the polynomial variable, POLY used as above returns the coefficients in unsimplified forms, e.g.

\[
'A*X+A*X' \ 'X' \ POLY \rightarrow \ '(A+A)*X'
\]

Some coefficients may be simplified by applying COLCT to the result, but this may also defeat the polynomial ordering. To provide better additional simplification, POLY accepts a list of names for its argument. The first argument is polynomialized with respect to the first name in the list; each coefficient is polynomialized with respect to
the second name; and so forth for each name in the list. Thus

\((X+A)^3 + (X+A^2)^2\) 'X' POLY

\[ (X^3 + (1+3*A)*X^2 + (2*A^2+3*A^2)*X + (A^2^2+A^3)') \]

\((X+A)^3 + (X+A^2)^2\) \{ X A \} POLY

\[ (X^3 + (3*A+1)*X^2 + 5*A^2*X + (A^4+A^3)') \]

This strategy may not be helpful if the result contains a fraction, since it will be polynomialized with respect to the second name, which counteracts the polynomialization with respect to the first.

### 17.4.1 Polynomial Programs

#### PADD

**Polynomial Add**

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_n } { P'_M }</td>
<td>{ P''_{max(M,N)} }</td>
<td></td>
</tr>
</tbody>
</table>

<< DUP2 SIZE SWAP SIZE
IF \< THEN SWAP END
OBJ~ DUP 2 + ROLL OBJ~ DUP
2 + ROLL
\- nshort nlong
<< 1 nshort
FOR n
nshort 1 + ROLL +
nshort ROLLD
NEXT
nlong \-LIST
>>

Determine the lists' sizes.
Put longer list in level 1.
Dump lists on stack.
Save list sizes.
Add each pair of terms:
Recombine into a list.

#### PSUB

**Polynomial Subtract**

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_n } { P'_M }</td>
<td>{ p''_{max(M,N)} }</td>
<td></td>
</tr>
</tbody>
</table>

<< PNEG PADD
>>

Negate and add.
### PNEG
**Polynomial Negate**

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_N }</td>
<td>{ -P_N }</td>
</tr>
</tbody>
</table>

\(\text{<< OBJ} \rightarrow \rightarrow n\)
\(\text{<< 1 } n\)
\(\text{START NEG n ROLL}\)
\(\text{NEXT}\)
\(n - \text{LIST}\)

Explode list, save count.
Negate each term.
Recombine list.

### PMUL
**Polynomial Multiplication**

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_N }</td>
<td>{ P'_M }</td>
<td>{ P''_{N+M} }</td>
</tr>
</tbody>
</table>

\(\text{<< DUP SIZE ROT DUP SIZE}\)
\(\text{IF DUP 4 PICK } >\)
\(\text{THEN 4 ROLL 4 ROLL}\)
\(\text{END } \rightarrow \text{ list1 } n1 \text{ list2 } n2\)
\(\text{<< {} } 1 \text{ n2}\)
\(\text{FOR } m\)
\(0 +\)
\(\text{list2 } m \text{ GET } \rightarrow \text{ mt}\)
\(\text{<< list1 OBJ} \rightarrow \text{ 1 SWAP}\)
\(\text{START mt } \star \text{ n1 ROLL NEXT}\)
\(n1 - \text{LIST PADD}\)
\(\text{NEXT}\)

Get list sizes.
Put longer list in level 4.
Start empty result list.
For each element of list2:
“Multiply” the result list by \(x^1\).
Get the \(m\)th element of list2.
Multiply by each list1 element.
Add products to the result list.
17.2 Expression Manipulations

**PDIVD**

*Polynomial Divide* 4D4C

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 3</th>
<th>level 2</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ \textit{P}_N }</td>
<td>{ \textit{P'}_M }</td>
<td>\Rightarrow</td>
<td>{ \textit{quotient} }</td>
<td>{ \textit{remainder} }</td>
</tr>
</tbody>
</table>

\(<\ DUP2\ SIZE\ SWAP\ SIZE\ >\\ 
\text{IF}\ OVER\ -\ DUP\ 0\ <\ THEN\ DROP2\ \{0\}\ 3\ ROLLD\\ 
\text{ELSE}\ SWAP\ ROT\ DUP\ 1\ GET\\ 
\quad n\ d\ t\\ 
\quad \Rightarrow\ \{\\}\ 3\ ROLLD\\ 
0\ SWAP\ START\\ 
\quad DUP\ 1\ GET\ \text{t}/\\ 
\quad ROT\ OVER\ +\\ 
\quad 3\ ROLLD\ 1\ n\\ 
\quad FOR\ m\\ 
\quad \quad OVER\ m\ GET\\ 
\quad \quad d\ m\ GET\\ 
\quad \quad 3\ PICK\ \ast\ ROT\\ 
\quad \quad m\ ROT\ PUT\ SWAP\ NEXT\\ 
\quad DROP2\ OVER\ SIZE\\ 
\quad MIN\ 1E499\ SUB\ NEXT\ d\\ 
\quad >>\\ 
\quad END\ >>

**PEXPR**

*Polynomial Expression* F240

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ \textit{P}_N } \quad x</td>
<td>\Rightarrow \quad \text{\textit{P}_N(x)}'</td>
</tr>
</tbody>
</table>

\(<\ \rightarrow\ x\ >\\ 
\text{\textless\ OBJ\textgreater\ >\\ 
\quad 0\ SWAP\ 1\\ 
\quad \text{FOR}\ n\ n\ 1\ +\ \text{ROLL}\\ 
\quad \quad x\ n\ 1\ \uparrow\ \ast\ +\\ 
\quad \quad -1\ \text{STEP}\\ 
\quad >>\ >\ >\ >

-582-
Expression Manipulations

**PPWR Polynomial Power**

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_N }</td>
<td>n</td>
<td>( \varphi )</td>
</tr>
</tbody>
</table>

\[ \langle \text{CASE} \rangle \]

- DUP 0 == THEN DROP2 \{ 1 \} END
- DUP 1 == THEN DROP END
- OVER SIZE 1 ==
  - THEN SWAP 1 GET
  - SWAP ^ 1 \rightarrow \text{LIST END}
- OVER SIZE 1 - ROT DUP 1 GET
- SWAP 2 1E499 SUB
- 4 ROLLD \rightarrow n N aN

\[ \langle \{ 0 \} \rangle \text{ SWAP} \]

- 0 n
- FOR i
  - DUP i PPWR
  - n i COMB
  - aN n i - ^ *
  - 1 \rightarrow \text{LIST PMUL}
  - ROT
  - IF 'i\#0'
    - THEN 1 N START 0 + NEXT END
  - PADD SWAP
  - NEXT DROP

\[ \langle \rangle \]

- END

\[ \langle \rangle \]

Check for simple cases:
- \( n=0 \).
- \( n=1 \).

*constant* case.

Get first coefficient \( a_N \).

"Rest" of list \( \{ P_{N-1} \} \).

Leave \( \{ P_{N-1} \} \) on stack.

Initialize a result polynomial.

For 0 to \( n \):

Compute \( P_{N-1}^j \).

\[ nC_i \]

\[ nC_i a_N^{n-i} \]

\[ nC_i a_N^{n-i} P_{N-1}^j \]

Raise power of old sum by \( N \).

Add new polynomial.

Return \( P_N^n \).
## Polynomial to List

<table>
<thead>
<tr>
<th>PLIST</th>
<th>Polynomial to List</th>
<th>BFED</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 2</td>
<td>level 1</td>
<td>level 1</td>
</tr>
<tr>
<td>'f(x)'</td>
<td>'x'</td>
<td>( { P_N } )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
<< & \rightarrow x \\
<< & \text{CASE} \\
& \{ 6 \ 7 \ 9 \ 0 \ 1 \ 13 \} \\
& \text{OVER \ TYPE \ POS \ DUP} \\
& \{ 6 \ 7 \} \ x \ \text{TYPE \ POS \ AND \ NOT} \\
& \text{THEN \ DROP} \ 
\begin{align*}
& \text{if} \ f(x) \ \text{a legal object type?} \\
& \text{Is} \ f(x) \ \text{a legal object type?} \\
& \text{And is} \ x \ \text{a global or local name?} \\
& \text{If not, then Bad Argument Type.} \\
& \text{If} \ f \ \text{is a number or unit object,} \\
& \text{return as zero-order.} \\
& \text{If} \ f \ \text{is a name,} \\
& \text{then if it's the same as} \ x, \\
& \text{then return as first order;} \\
& \text{otherwise as zero-order.} \\
& \text{Get top-level function and arguments.} \\
& \text{If a polynomial function,} \\
& \text{then polynomialize one argument.} \\
& \text{If two arguments,} \\
& \text{then polynomialize the other.} \\
& \text{If it's not /,} \\
& \text{then execute the polynomial operation.} \\
& \text{For division, compute the fraction,} \\
& \text{and add the remainder.} \\
& \text{Discard copy of} \ f(x). \\
& \text{Return} \ f(x) \ \text{as zero order.} \\
\end{align*}
\end{align*}
\]

\[
\begin{align*}
& \text{Save polynomial variable name.} \\
& \text{Save polynomial variable name.} \\
& \text{Is} \ f(x) \ \text{a legal object type?} \\
& \text{Is} \ f(x) \ \text{a legal object type?} \\
& \text{And is} \ x \ \text{a global or local name?} \\
& \text{And is} \ x \ \text{a global or local name?} \\
& \text{If not, then Bad Argument Type.} \\
& \text{If not, then Bad Argument Type.} \\
& \text{If} \ f \ \text{is a number or unit object,} \\
& \text{If} \ f \ \text{is a number or unit object,} \\
& \text{return as zero-order.} \\
& \text{return as zero-order.} \\
& \text{If} \ f \ \text{is a name,} \\
& \text{If} \ f \ \text{is a name,} \\
& \text{then if it's the same as} \ x, \\
& \text{then if it's the same as} \ x, \\
& \text{then return as first order;} \\
& \text{then return as first order;} \\
& \text{otherwise as zero-order.} \\
& \text{otherwise as zero-order.} \\
& \text{Get top-level function and arguments.} \\
& \text{Get top-level function and arguments.} \\
& \text{If a polynomial function,} \\
& \text{If a polynomial function,} \\
& \text{then polynomialize one argument.} \\
& \text{then polynomialize one argument.} \\
& \text{If two arguments,} \\
& \text{If two arguments,} \\
& \text{then polynomialize the other.} \\
& \text{then polynomialize the other.} \\
& \text{If it's not /,} \\
& \text{If it's not /,} \\
& \text{then execute the polynomial operation.} \\
& \text{then execute the polynomial operation.} \\
& \text{For division, compute the fraction,} \\
& \text{For division, compute the fraction,} \\
& \text{and add the remainder.} \\
& \text{and add the remainder.} \\
& \text{Discard copy of} \ f(x). \\
& \text{Discard copy of} \ f(x). \\
& \text{Return} \ f(x) \ \text{as zero order.} \\
& \text{Return} \ f(x) \ \text{as zero order.}
\end{align*}
\]

- 584 -
## LPWR (List to Power)

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ P_N }</td>
<td>{ n }</td>
<td>( P_N^n )</td>
</tr>
</tbody>
</table>

```
<< CASE DUP SIZE 1 ≠
  THEN DROP SWAP END
1 GET DUP TYPE
  THEN DROP SWAP END
DUP 0 < THEN DROP SWAP END
DUP FP THEN DROP SWAP END
PPWR
END
>>
```

- Power not zero-order.
- Power not a real number.
- Negative power.
- Fractional power.
- Do the involution.

## POLY (Polynomialize Expression)

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'f(x)'</td>
<td>'x'</td>
<td>'P_N(x)'</td>
</tr>
<tr>
<td>'f(x)'</td>
<td>{ names }</td>
<td>'P_N(x)'</td>
</tr>
</tbody>
</table>

```
<< IF DUP TYPE 5 SAME
  THEN OBJ- 1 -
ELSE 0
END -LIST - x rest
<< x PLIST
  IF rest SIZE
    THEN OBJ- \rightarrow n
<< 1 \rightarrow n
    START
      rest POLY n ROLL
    NEXT
    n -LIST
  >>
END
x PEXPR
>>
```

- If the 2nd argument is a list, then explode it.
- Otherwise, “rest” is null.
- Save first name, and rest.
- Polynomialize with respect to x.
- If rest is non-null:
  - Then explode the coefficient list, and for each term,
  - Polynomialize against rest list.
- Recombine coefficient list.
- Convert list to expression.
18. Calculus

The HP 48's mathematical capabilities extend into the realm of the calculus, including differentiation, integration, and polynomial approximations. The derivative function \( \partial \) can symbolically differentiate expressions containing almost any combination of HP 48 functions for which sensible derivatives exist (see section 3.1). By applying the derivative repeatedly, the TAYLR command generates Maclaurin series for arbitrary expressions, which can be generalized into Taylor's polynomials at any point. Computing antiderivatives is a more difficult problem, but the HP 48 can symbolically integrate a fixed set of common integrand patterns. For integrals that do not match the built-in patterns, the HP 48 can still compute accurate numerical integrals.

Although summation might not strictly be considered as part of the calculus, we will discuss summations as a preliminary to the treatment of integration. Besides the close mathematical association between summation and integration, the HP 48 functions \( \Sigma \) and \( \int \) have a number of similar properties.

18.1 Differentiation

The derivative function \( \partial \) is quite straightforward to use, except that you must choose whether to carry out a chain-rule derivative in steps or all at once. In either case, you have to identify

a. the expression to be differentiated, and  
b. the variable of differentiation.

Since \( \partial \) is a function, you can specify these items as two RPN stack arguments, or as arguments for \( \partial \) within an expression. This choice of RPN or expression format also determines whether the differentiation is performed in a single operation (RPN) or one step at a time (expression).

To use \( \partial \) as a stack command, you enter the expression to be differentiated into level 2, and the name of the differentiation variable into level 1, then execute \( \partial \). The result is an expression representing the derivative of the original expression. It is fully differentiated--the \( \partial \) function does not appear in the result.

The expression syntax for a derivative does not follow the normal HP 48 convention of parenthesizing stack arguments, where \( A B \partial \) should become '\( \partial(A,B) \)'. Instead, the form of \( \partial \) within expressions is modeled after the standard written form \( D_B A \) (derivative of \( A \) with respect to \( B \)): this derivative is expressed in HP 48 syntax as '\( \partial B(A) \)'. The parentheses are necessary to separate the expression \( A \) from the name \( B \), and from the
remainder of the expression, if any. The more common ratio form is realized in the EquationWriter presentation of derivatives (section 6.7.3.3), where the differentiation variable can be shown as a denominator with the prefix $\partial$:

$$\frac{\partial f}{\partial B}(A)$$

The HP 48 uses the symbol $\partial$ rather than $d$ or $D$ to leave the ordinary letters available for names. $dA/dB$ is perhaps a more common written form than $DB A$, but given the general HP 48 rules for naming variables, $'dA/dB'$ could also mean $dA$ divided by $dB$, so this form is not used in the linear format.

$\partial$ also differs from other HP 48 functions in that it executes differently as a stack command than it does when it is part of an expression. In the RPN case, the derivative is repeatedly executed until the $\partial$ function is no longer present. However, when an expression containing $\partial$ is evaluated, the derivative is only carried out as far as a single application of the chain rule of differentiation, which is:

$$\frac{df}{dx}(g(x)) = \frac{df}{dg} \cdot \frac{dg}{dx}.$$

For example, to compute $\frac{d}{dx} \sin(\cos(x))$, multiply

$$\frac{df}{dg} = \frac{d}{d(\cos(x))} \sin(\cos(x)) = \cos(\cos(x))$$

by

$$\frac{dg}{dx} = -\sin(x),$$

to obtain the result

$$-\cos(\cos(x)) \sin(x).$$
On the HP 48, you can watch this calculation unfold by entering

\[ \text{RAD} \quad \partial X(\sin(\cos(X)))', \]

then

\[ \text{EVAL} \quad \rightarrow \quad '\cos(\cos(X)) \ast \partial X(\cos(X))' \]
\[ \text{EVAL} \quad \rightarrow \quad '\cos(\cos(X)) \ast (-\sin(X) \ast \partial X(X))' \]
\[ \text{EVAL} \quad \rightarrow \quad '\cos(\cos(X)) \ast (-\sin(X))'. \]

Each EVAL applies the chain rule through one level. If you are not interested in the intermediate result, you can obtain the final result in one step:

\[ '\sin(\cos(X))' \quad 'X' \quad \partial \quad \rightarrow \quad '\cos(\cos(X)) \ast (-\sin(X))'. \]

The one-step derivative obtained by including \( \partial \) within an expression is consistent with the general flavor of HP 48 expression evaluation (section 3.5.2), which substitutes values for each variable in an expression, but does not recursively evaluate the substituted subexpressions. This style of differentiation is quite useful as a teaching tool, whereby you can observe results of successive applications of the chain rule.

### 18.1.1 Calculus with Trigonometric Functions

If you differentiate expressions containing trigonometric functions while the HP 48 is in degrees or grads mode, factors of \( \pi/180 \) or \( \pi/200 \) will appear in the result:

\[ \text{DEG} \quad '\sin(X)' \quad 'X' \quad \partial \quad \rightarrow \quad '\cos(X) \ast (\pi/180)' \]

This is correct mathematically, but it may surprise you when you take a derivative without thinking about the angle mode. In effect, any trigonometric function is a different function in each angle mode. For example, in radians mode, \( \sin \) corresponds directly to the usual sine function, where the argument is expressed in radians. However, in degrees mode, \( \sin(x) \) is actually a representation of \( \sin(180x/\pi) \). Differentiating the latter expression returns \( (180/\pi) \cos(180x/\pi) \).

The same reasoning applies to integrals. For example, if you compute (see section 18.4.2)

\[ \int_{0}^{\pi} \sin x \, dx \]

- 589 -
in radians mode, you obtain 2, as you might expect. But in degrees mode:

\[ \int(0,180,\sin(X),X) \rightarrow \text{NUM} \rightarrow 114.591559026 \]

The difference in the two results is the factor of \( 180/\pi \) that arises in the integral of \( \sin \) in degrees mode. The result is easy to understand if you plot \( \sin \) between 0 and 180:

![Graph of \( \sin \) between 0 and 180](image)

Even a quick glance at the picture shows that the area under the curve is certainly closer to 115 than to 2.

### 18.1.1.1 User-Defined Derivatives

Almost all HP48 functions that represent continuous and differentiable functions have derivatives included in their built-in definitions. The percent functions \%, \%CH, and \%T are exceptions, having no built-in derivatives. Attempting to differentiate an expression containing any of these functions yields a strange-looking result:

\[ \text{"%T}(X,Y) \ 'X' \ \partial \ \rightarrow \ \text{"der}%T((X,Y,1,0)) \]

This is an example of the HP 48's provision for *user-defined derivatives*, which are derivatives that you can define to substitute for missing derivatives. When the function \( \partial \) encounters a function \( \phi \) for which no derivative is available, it does not error. Instead, it evaluates the global name \( \text{der}\phi \), created by appending the function name \( \phi \) to the letters "der." If the corresponding variable exists, then the stored object, which should be a user-defined function (section 8.5), is executed. Otherwise, the unevaluated symbolic form of the function is returned, as in the example above. This strategy applies to:

- Built-in or library functions with no associated derivatives, such as the percent functions, or discontinuous functions like \( \text{IP} \) and \( \text{MOD} \).
- User-defined functions for which no definition has been supplied.
User-defined functions with programs as their defining objects (section 8.5.3).

The value of this approach is that it allows you to continue with calculations that might otherwise be stalled, by supplying the missing derivatives. If you define the necessary dervs before differentiating, you will not see them in the results. But you can also wait until one appears in a result, then store a definition for it and evaluate the result to compute the derivative. For example, the following creates a user-defined derivative for %T:

\[ \text{'der\%T(x,y,dx,dy) = (dy/x - y/(x^2) * dx)*100'} \quad \text{DEFINE} \]

With this definition, you can evaluate the previous result:

\[ \text{'der\%T(X,Y,1,0)'} \quad \text{EVAL} \quad \text{''-(Y/X^2*100).'} \]

(where neither X and Y has a current value).

In general, the user-defined derivative of a function of n arguments requires 2n arguments. The first n of these are the arguments of the original function, and the second n are the derivatives of the first n arguments with respect to the differentiation variable. These latter arguments are necessary for chain-rule differentiation. You can see the argument structure in this symbolic example:

\[ \text{''Oxd(y1,y2,-t ya) EVAL = 'derd(yiyz, ya,0x(y1),0x(y2),*-+,0x())'} \]

T% is an example of a two-argument function. It is defined as

\[ T%(x,y)=100\frac{y}{x}, \]

and its derivative with respect to z is

\[ \frac{dT\%}{dz}(x,y) = \frac{dT\%}{dx}(x,y)\frac{dy}{dz} + \frac{dT\%}{dy}(x,y)\frac{dx}{dz}. \]

dT%/dx is \(-100y/x^2\), and dT%/dy is \(100/x\). The presence of the factors dx/dz and dy/dz necessitates the extra two arguments for derT%. The easiest way to write derT% as a user-defined function is to name the four arguments x, y, dx, and dy, and then write the defining expression like the total differential of T%, i.e.

\[ \frac{dT\%}{dx}\, dx + \frac{dT\%}{dy}\, dy = 100(-\frac{y}{x^2}\, dx + \frac{dy}{x}) \]
Simplifying this a little, and converting it to HP48 syntax leads to the definition for\nder%T given above.

User-defined derivatives are not necessary for user-defined functions that are defined with algebraic objects. Differentiating such functions automatically carries the differentiation through their internal definitions. For example, if you use the following to create a user-defined function for the secant:

\[
'\text{SEC}(x) = \text{INV}(\text{COS}(x))' \quad \text{DEFINE},
\]

then you can differentiate expressions containing SEC:

\[
\text{RAD} '\text{SEC}(\theta)' '\theta' \partial \Leftrightarrow '\text{SIN}(\theta)/\text{COS}(\theta)^2'.
\]

The derivative of sec(θ) can also be expressed as tanθ/cosθ. You can write a derivative for SEC that reflects this form:

\[
'der\text{SEC}(x, dx) = \text{TAN}(x)/\text{COS}(x) * dx * (3.14159265359/\text{ACOS}(-1))' \quad \text{DEFINE}.
\]

(The factor of \(\pi/\cos^{-1}(-1)\) provides for differentiation in any angle mode.) However, \(\partial\) only uses user-defined derivatives when no default is available, so it will not use derSEC in differentiating SEC. If you want to force the use of derSEC, you must rewrite SEC with a program as its defining object:

\[
<< \to x \<< x \ \text{COS} \ \text{INV} >> >> '\text{SEC'} \ \text{STO}
\]

For example, with these definitions, and 'θ^2' stored in the variable X, then

\[
'\text{SEC}(X)' '\theta' \partial \Leftrightarrow '\text{TAN}(\theta^2)/\text{COS}(\theta^2)*(2*\theta)'.
\]

### 18.1.2 Formal Derivatives

Executing \(\partial\) as an RPN command may cause repeated evaluation of the differentiated expression. If the differentiation variable has a value (anywhere in the current path), the variable will disappear from the result. For example, with 0 stored in X,

\[
'\text{SIN}(\text{COS}(X))' 'X' \partial \Leftrightarrow 0,
\]

which is not too helpful if you are looking for a symbolic result.

One simple way to ensure a symbolic result is to purge the variable of differentiation
before executing $\partial$. However, this is undesirable when you need to keep its stored value for further calculations. You might save the value, purge the variable, differentiate, and then restore the variable, but there may be one or more variables with the same name in parent directories, so that you would have to find and treat all of them the same way.

The program FDER listed below illustrates a neat solution to this problem. It uses \texttt{\texttt{MATCH}} (section 17.3) to substitute a local variable for the differentiation variable, then differentiates with respect to that local variable. The local variable is given its own name as its value, so that repeated evaluation of that name during differentiation never does anything except return the same name (with a global variable, this trick would lead to endless execution--see section 3.6.1). After differentiation, the original global name is substituted back for the local name. Using this program in place of $\partial$ always produces a result expressed in terms of the differentiation variable:

$$'\sin(\cos(x))' 'x' \texttt{FDER} \rightarrow '\cos(\cos(x))' * -'\sin(x)'$$

<table>
<thead>
<tr>
<th>FD ER</th>
<th>Formal Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{level 2}</td>
<td>\texttt{level 1}</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>$'$ $x'$</td>
</tr>
</tbody>
</table>

\begin{verbatim}
<< DUP - x \t 4. \\
<< x SHOW Make references to $x$ explicit. \\
x \{ \t 4. \} + MATCH DROP Replace original variable with \t 4. \\
\'\t 4.\' DUP STO \t 4. will return \t 4. \\
\t 4. \partial Differentiate. \\
x \'\t 4.\' STO EVAL \t 4. will return \t 4. \\
>> Restore original variable.
\end{verbatim}

The only flaw in FDER is that you can obtain an incorrect result if the differentiated expression already contains the local name used by FDER. To minimize this possibility, FDER and other programs listed later in this chapter use the unusual local name \t 4. This name has no mnemonic value, is obviously not used in common mathematical exercises, and gives a poor appearance to expressions containing it. If by some chance you do like to use this (local) name in your calculations, you can always rewrite FDER using any other name you prefer.
18.2 Taylor’s Polynomials

The Nth degree Taylor’s polynomial for a function \( f(x) \) at the point \( x = x_0 \) is defined by:

\[
f(x) = \sum_{n=0}^{N} \frac{(x-x_0)^n}{n!} \frac{d^n}{dx^n} f(x) \bigg|_{x=x_0}
\]

The special case of \( x_0 = 0 \) is called MacLaurin’s formula:

\[
f(x) = \sum_{n=0}^{N} \frac{x^n}{n!} \frac{d^n}{dx^n} f(x) \bigg|_{x=0}
\]

These definitions are valid for functions for which all derivatives of \( f \) exist up to degree \( n \). For \( N = \infty \), the polynomial is equal to the function \( f \). For finite \( N \), the polynomial constitutes an approximation to the function; the higher the degree, the better the approximation.

The TAYLR command computes the Nth degree Taylor’s polynomial for a function at the origin (MacLaurin’s formula). To use TAYLR, you enter an expression for the function in level 3, the polynomial variable name in level 2, and the polynomial degree in level 1.

**Example.** Compute the fifth-order Taylor’s polynomial for \( \sin x \) at \( x = 0 \).

```plaintext
RAD 'SIN(X)' 'X' 5 TAYLR = 'X-1/3!*X^3+1/5!*X^5'
```

In this result, the even-degree terms are absent, because they are all proportional to \( \sin^n 0 = 0 \).

To produce a Taylor’s polynomial at a point \( x_0 \) other than the origin, it is only necessary to make a translation of the coordinate system such that \( x = x' + x_0 \), use TAYLR with the variable \( x' \), then translate the system back by substituting \( x' = x-x_0 \). The program TAYLRX0 performs these operations. TAYLRX0 uses the same input arguments as TAYLR, with an additional argument to specify the point \( x_0 \).

**Example.** Compute the 3rd degree Taylor’s polynomial for \( \sin x \) at the point \( x = \pi/2 \).

```plaintext
RAD 'SIN(X)' 'X' '\pi/2' 3 TAYLRX0 = '1-.5*(X-\pi/2)^2'.
```

Note that the result is the same as the 3rd degree Taylor’s polynomial for \( \cos (x-\pi/2) \), which follows from the identity \( \sin x = \cos (x-\pi/2) \).
### Taylor's Polynomial at $x_0$

<table>
<thead>
<tr>
<th>level 4</th>
<th>level 3</th>
<th>level 2</th>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'expression'</td>
<td>'name'</td>
<td>$x_0$</td>
<td>degree</td>
<td>$\varphi$</td>
</tr>
</tbody>
</table>

- << RCLF ← x x0 n f  
  - Save the name, $x_0$, degree, flags.
- << -3 CF  
  - Symbolic evaluation.
- x SHOW  
  - Make all instances of $x$ explicit.
- x 'x+x0' EVAL 2 →LIST  
  - Substitution patterns.
- tMATCH DROP  
  - Substitute $x + x_0$ for $x$.
- x n TAYLR  
  - Make the expansion.
- x 'x-x0' EVAL 2 →LIST  
  - Reverse substitution pattern.
- tMATCH DROP  
  - Substitute $x-x_0$ for $x$.
- f STOF  
  - Restore flags.

$t_0$ may be a name, expression, or number.

You can only apply TAYLR meaningfully to functions for which the function itself and its derivatives up to the $N$th-order are defined at $x = 0$. For example, you cannot compute a polynomial for $\sqrt{x}$, since its first derivative is proportional to $1/\sqrt{x}$, which is infinite at $x = 0$.

### 18.3 Summations

The summation function $\Sigma$ provides an automated means for executing definite sums

$$\sum_{index=start}^{stop} summand$$

Used as an RPN command, $\Sigma$ uses four arguments:

- $index$ start stop summand $\Sigma \varphi$ sum.

$Index$ should be a global or local name, and $start$, $stop$, and $summand$ can be algebraic objects or other objects allowed in expressions. Normally, $summand$ contains at least one use of the $index$ name.

When $\Sigma$ itself is embedded in an expression, the syntax is

$$\Sigma(index=start,stop,summand)$$

When $\Sigma$ is evaluated, it returns one of the following:
• The original sum, unchanged, if either of the limits are symbolic;
• A symbolic sum of terms, if both limits evaluate to numbers and the summand contains symbolic arguments other than the index; thus
  \[ \sum_{i=1}^{3} (1+A) \] EVAL \[ \Rightarrow '1+A+(2+A)+(3+A)'. \]
• A numeric sum, if the limits and the summand all evaluate to numbers; thus
  \[ \sum_{i=1}^{100} (1+2) \] EVAL \[ \Rightarrow 338350 \]

If you differentiate a sum, the derivative is just pushed inside of the summation:

\[ \sum_{i=A}^{B} F(X,i) \] 'X' \[ \Rightarrow \sum_{i=A}^{B} \partial X(F(X,i)) \]

The derivative ignores the limits, which are presumed to be integer functions with no continuous derivative. Differentiating with respect to the index always returns zero.

Similar considerations apply to the integration of a sum. When the integral is evaluated, if the summand is an integrable pattern, the result is the (unevaluated) sum with the summand replaced by its definite integral:

\[ \int_{A}^{B} \sum_{i=1}^{M} \sin(N*X), X \] EVAL EVAL \[ \Rightarrow \sum_{i=1}^{M} \left( \frac{-\cos(N*B)}{N} \right) - \sum_{i=1}^{M} \left( \frac{-\cos(N*A)}{N} \right) \]

This is more legible in EquationWriter form as

\[ \int_{A}^{B} \sum_{i=1}^{M} \sin(N*X) \text{ EVAL EVAL} \Rightarrow \sum_{i=1}^{M} \left( \frac{\cos(N*B)}{N} \right) - \sum_{i=1}^{M} \left( \frac{\cos(N*A)}{N} \right) \]

If the summand is not integrable, the integral is not pushed inside the sum; the result retains the sum as the integrand.

18.3.1 Summation Patterns
The HP 48 pattern-matching capabilities (section 17.3) may be used to develop a summation processor similar to the integral command \( \int \) (section 18.4), which matches its integrands against a table of pre-defined patterns. For example, a simple sum of integers evaluates to

\[ \sum_{i=1}^{n} i = n(n+1)/2. \]

In HP 48 pattern-matching terms, any sum of the form \( \sum_{i=1}^{n} &1 \) can be replaced by \( &2 *( &2 +1)/2 \), assuming \( &2 \) is an integer.
The program XSUM below takes any algebraic object and tests its sums against a small database of patterns stored in the variable $\Sigma$PATTERNS. If a match is found, the corresponding sum is replaced with a formula from the database. The $\Sigma$PATTERNS listed below contains replacements for the sum of integers, squared integers, and cubed integers. You can add additional pattern/replacement pairs to $\Sigma$PATTERNS as you please.

$$\sum(l=1,N,l) \quad \text{XSUM} \iff 'N*(N+1)/2'$$

The program INTMATCH in section 18.4.1.3 is similar in spirit to XSUM, but it uses a more careful approach that permits more flexible matching than XSUM. The methods used by XINT can easily be adapted to summation matching.
18.4 Integration

Unfortunately, there is no analog of the chain rule of differentiation that allows you to compute the integral of an arbitrary expression directly from the antiderivatives of the functions in the expression. General-purpose symbolic integration algorithms require considerable memory resources, so the HP 48 limits itself to integration of a certain set of integrands, using its pattern-matching ability to generalize the set. For these and any other integrals, you also can invoke a sophisticated numerical integrator.

\( \int \) is an HP 48 function (section 3.1), so that it can be included in expressions or executed as an RPN command. The RPN usage for \( \int \) is as follows:

\[
\text{lower-limit upper-limit integrand name } \int \Rightarrow \text{ result},
\]

where \( \text{name} \) is a name object representing the variable of integration, and the other three arguments can be arbitrary expressions. The arguments appear in the same order in which they appear in the expression form of an integral:

\[
\int(\text{lower-limit, upper-limit, integrand, name})
\]

The single function \( \int \) combines both symbolic and numerical integration, with the choice of method determined implicitly by the current function execution mode (section 3.5.5.2). With flag -3 clear (symbolic execution), \( \int \) uses the pattern-matching system to return a symbolic result. With flag -3 set (numerical execution), \( \int \) uses a numerical algorithm to return a real number result. You can also obtain a numerical result regardless of the flag setting by using \(-\text{NUM}\) to evaluate \( \int \).

18.4.1 Symbolic Integration

Evaluation of \( \int \) in symbolic execution mode (flag -3 clear) searches the integrand for functions for which the HP 48 knows antiderivatives, and which are linear in the variable of integration.

- A function linear in a variable \( x \) has an argument of the form \( \pm ax \pm b \), where \( a \) and \( b \) are constants (relative to \( x \)). The HP 48 can integrate \( \sin(x) \), so therefore it can also integrate \( \sin(2x-6) \) or \( \sin(ax+z) \).
- Functions may be multiplied or divided by a constant, e.g. \( C\sin(x) \) or \( \sin(x)/(a+b) \).

Each such function is integrated and removed from the integrand. The final result is a sum of integrated terms plus an integral containing any remaining terms.

Note that symbolic integration (like \( \text{ISOL} \)--see section 16.4.1) does not attempt to find
indirect references in the integrand to the variable of integration. When the integrand contains variables that in turn contain expressions in the variable of integration (at any level), you should execute SHOW as a preliminary to \( \int \), to make explicit all possible references to the integration variable.

Symbolic integration is performed in two steps. The first step, which is performed by \( \int \), uses the where function \(|\) to show a formal result prior to evaluation at the limits. For example,

\[
\text{RAD } '\int(A,B,\cos(X),X)' \text{ EVAL}
\]

\[
eq '\sin(X)/\partial X(X) | (X=B) - (\sin(X)/\partial X(X) | (X=A))' .
\]

This result is more legible in the EquationWriter environment:

\[
\left. \frac{\sin(X)}{\frac{\partial}{\partial X}(X)} \right|_{X=B} - \left. \left( \frac{\sin(X)}{\frac{\partial}{\partial X}(X)} \right) \right|_{X=A}
\]

This is an expanded form of the common “double-where” notation used with integral results:

\[
\left. \frac{\sin(x)}{\frac{\partial}{\partial x}(x)} \right|_{x=B} - \left. \left( \frac{\sin(x)}{\frac{\partial}{\partial x}(x)} \right) \right|_{x=A}
\]

which represents the difference between the expression on the left evaluated at the upper limit and at the lower limit.

The derivative that appears in the denominator of this result computes the linear coefficient of the integration variable as it appears in the argument of the function that is matched. That is, if the current integrand were \( A*X+B \), the result would have a denominator of \( \partial X(A*X+B) \), which evaluates to \( A \). Using the derivative in this manner
is the most effective way to determine the linear coefficient.

The second step in symbolic integration is evaluation at the limits. Applying EVAL again to the last result returns

\[ \sin(b) - \sin(a) \]

in which the integration variable has been replaced by its values at the limits.

You may find that most commonly you execute EVAL twice without even bothering to view the intermediate result. The HP 48 preserves the two-step procedure for pedagogical purposes, and to permit manipulations before the final evaluation for cases where automatic evaluation might lead to errors (see section 21.3.9).

Note that \( f \) always computes a definite integral, requiring explicit limits of integration. You can obtain an indefinite integral by choosing formal (undefined name) limits, then discarding the term arising from the lower limit. The user-defined function INDEF automates this process:

\[
\text{INDEF} \quad \text{Indefinite Integral} \quad 4\text{CEE}
\]

<table>
<thead>
<tr>
<th>'integrand'</th>
<th>'name'</th>
<th>'integral'</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 2</td>
<td>level 1</td>
<td>level 1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\langle & f \ x \\
\langle & f \ x \ SHOW \\
\langle & 'x' \ DUP \ DUP2 \ STO \\
\langle & \text{ROT} \ 4 \ \text{PICK} \ f \\
\langle & \text{IF} \ \{ 'f(\&1,\&2,\&3,\&4)' \ x \} \ \text{IMATCH} \\
\langle & \text{THEN} \ \text{DROP} \ f \\
\langle & \text{ELSE} \ \text{OBJ-} \ 3 \ \text{DROPN} \\
\langle & \text{EVAL} \\
\langle & x \ \text{ROT} \ 2 \ -\text{LIST} \ \text{IMATCH} \ \text{DROP} \\
\langle & \text{END} \\
\langle & >>
\end{align*}
\]

INDEF takes two arguments, an integrand expression and the name of the variable of integration.

\[
\frac{x+1}{(1+x^2)} \quad 'x' \quad \text{INDEF} \quad '\arctan(x)+x^2/2'
\]
Because of its user-defined function structure, \texttt{INDEF} may also be evaluated within an expression. However, you should use \texttt{QUOTE} (section 16.7.1) to prevent evaluation of the arguments, e.g. \texttt{`INDEF(QUOTE(F(X)),QUOTE(X))'}.

### 18.4.1.1 Integration Patterns
The set of integrand terms that the HP 48 can actually integrate symbolically is listed in Table 18.1 below. This set was chosen to include the following:

- All of the built-in functions which have closed-form anti-derivatives expressible in terms of other built-in functions, except \texttt{LNP1}. Thus \texttt{ATAN} is included, but not \texttt{!} (factorial/gamma function)
- Derivatives of all built-in functions. For example, since the derivative of \texttt{ATAN(X)} with respect to \texttt{X} is given by the HP 48 as \texttt{INV(1+X^2)}, the latter pattern is included, along with some minor variants.
- Additional "common" patterns, such as \texttt{1/TANH(X)} and \texttt{1/(SIN(X)*COS(X))}.
- Summations using \texttt{\Sigma} (section 18.3).
- Derivatives.
- Linear combinations of the above patterns, where the coefficients are constants.

In a number of cases, the built-in integrator will fail to match a pattern that you may think is obvious, especially if it is formally equivalent to a pattern that can be integrated. For example, \texttt{INV(1+X^2)} integrates successfully to \texttt{ATAN(X)}, but \texttt{INV(1+SQ(X))} does not. The integral pattern matching has a certain amount of flexibility, but not enough to cover all permutations of a pattern that you can obtain with a chain of identity transformations. In many cases, you should try expanding, collecting, and otherwise reorganizing terms so that the integrator can find a good match. If you apply \texttt{EXPAN} to \texttt{INV(1+X^2)}, you obtain \texttt{INV(1+X*X)}, which still does not integrate. However, next using \texttt{COLCT} returns \texttt{INV(1+X^2)}, which is one of the recognized patterns. \texttt{POLY} (section 17.4) is also a good way to reduce a general polynomial to a form that can be integrated.

Table 18.1 lists the integrand patterns recognized by the HP 48, and their corresponding antiderivatives. In the table, \( \phi \) represents a linear function of the variable of integration; the listed antiderivatives should be divided by the first order coefficient in \( \phi \). Patterns marked with a \( \dagger \) work also when \( 1/(...) \) is replaced by \texttt{INV(...)}. 
### Table 18.1. HP 48 Symbolic Integral Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Antiderivative</th>
<th>Pattern</th>
<th>Antiderivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS((\phi))</td>
<td>(\phi \cdot ACOS(\phi) - \sqrt{1 - \phi^2})</td>
<td>TAN((\phi))</td>
<td>-LN(COS((\phi)))</td>
</tr>
<tr>
<td>ALOG((\phi))</td>
<td>0.434294481904*ALOG((\phi))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASIN((\phi))</td>
<td>(\phi \cdot ASIN(\phi) + \sqrt{1 - \phi^2})</td>
<td>TAN((\phi))/COS((\phi))</td>
<td>INV(COS((\phi)))</td>
</tr>
<tr>
<td>ATAN((\phi))</td>
<td>(\phi \cdot ATAN(\phi) - LN(1 + \phi^2)/2)</td>
<td>1/TAN((\phi)) +</td>
<td>LN(SIN((\phi)))</td>
</tr>
<tr>
<td>COS((\phi))</td>
<td>SIN((\phi))</td>
<td>1/TAN((\phi)) +</td>
<td>-INV(SIN((\phi)))</td>
</tr>
<tr>
<td>1/(COS((\phi))*SIN((\phi)))</td>
<td>LN(TAN((\phi)))</td>
<td>TANH((\phi))</td>
<td>LN(COSH((\phi)))</td>
</tr>
<tr>
<td>COSH((\phi))</td>
<td>SINH((\phi))</td>
<td>TANH((\phi))/COSH((\phi))</td>
<td>INV(COSH((\phi)))</td>
</tr>
<tr>
<td>1/(COSH((\phi))*SINH((\phi)))</td>
<td>LN(TANH((\phi)))</td>
<td>1/TANH((\phi)) +</td>
<td>LN(SINH((\phi)))</td>
</tr>
<tr>
<td>1/(COSH((\phi))^2)</td>
<td>TANH((\phi))</td>
<td>1/(TANH((\phi)) +</td>
<td>-INV(SINH((\phi)))</td>
</tr>
<tr>
<td>EXP((\phi))</td>
<td>EXP((\phi))</td>
<td>V(\phi)</td>
<td>2*(\phi)^1.5/3</td>
</tr>
<tr>
<td>EXPM((\phi))</td>
<td>EXP((\phi)) - (\phi)</td>
<td>1/V(\phi)</td>
<td>2*V(\phi)</td>
</tr>
<tr>
<td>LOG((\phi))</td>
<td>0.434294481904*LOG((\phi))</td>
<td>1/(2*V((\phi))/V(\phi))</td>
<td>2*V((\phi))/V(\phi) ^ 0.5</td>
</tr>
<tr>
<td>SIGN((\phi))</td>
<td>ABS((\phi))</td>
<td>(\phi^Z) (Z symbolic)</td>
<td>IFTE((Z = -1, LN(\phi), \phi^\phi)^(Z+1)/(Z+1))</td>
</tr>
<tr>
<td>SIN((\phi))</td>
<td>-COS((\phi))</td>
<td>(\phi^Z) (Z real, (\neq 0, -1))</td>
<td>(\phi^\phi)</td>
</tr>
<tr>
<td>1/(SIN((\phi))*COS((\phi)))</td>
<td>LN(TAN((\phi)))</td>
<td>(\phi^0)</td>
<td>(\phi)</td>
</tr>
<tr>
<td>1/(SIN((\phi))*TAN((\phi)))</td>
<td>-INV(SIN((\phi)))</td>
<td>(\phi^-1)</td>
<td>LN((\phi))</td>
</tr>
<tr>
<td>1/(SIN((\phi))*TAN((\phi)))</td>
<td>-INV(SIN((\phi)))</td>
<td>1/(\phi) +</td>
<td>LN((\phi))</td>
</tr>
<tr>
<td>1/(SIN((\phi))^2)</td>
<td>-INV(TAN((\phi)))</td>
<td>1/(1-(\phi^2)) +</td>
<td>ATANH((\phi))</td>
</tr>
<tr>
<td>SINH((\phi))</td>
<td>COSH((\phi))</td>
<td>1/(1+(\phi^2))</td>
<td>ATANH((\phi))</td>
</tr>
<tr>
<td>1/(SINH((\phi))^2)</td>
<td>-INV(TANH((\phi)))</td>
<td>1/((\phi^2+1))</td>
<td>ATANH((\phi))</td>
</tr>
<tr>
<td>1/(SINH((\phi))*COSH((\phi)))</td>
<td>LN(TANH((\phi)))</td>
<td>1/((\sqrt{(\phi-1)}*\sqrt{(\phi+1)}))</td>
<td>ACOSH((\phi))</td>
</tr>
<tr>
<td>1/(SINH((\phi))*TANH((\phi)))</td>
<td>-INV(SINH((\phi)))</td>
<td>1/(\sqrt{1-(\phi^2)})</td>
<td>ASIN((\phi))</td>
</tr>
<tr>
<td>SQ((\phi))</td>
<td>(\phi^-3/3)</td>
<td>1/(\sqrt{1+(\phi^2)})</td>
<td>ASINH((\phi))</td>
</tr>
<tr>
<td>TAN((\phi))^2</td>
<td>TAN((\phi)) - (\phi)</td>
<td>1/(\sqrt{(\phi^2+1)})</td>
<td>ASINH((\phi))</td>
</tr>
</tbody>
</table>

### 18.4.1.2 Derivative and Integral

The derivative of an integral, in the most general case where the limits and the integrand are functions of the differentiation variable, is given by:

\[
\frac{d}{dx} \int_{i(x)}^{u(x)} f(x,t) dt = f(x,u(x)) \frac{du}{dx} - f(x,1(x)) \frac{dl}{dx} + \int_{i(x)}^{u(x)} \frac{d}{dy} f(x,t) dt
\]
In HP 48 terms, this definition translates to:

\[ \int (L(X), U(X), F(X, T), T) \]  
\[ \quad \partial X \]  
\[ = F(U(X)) \cdot \text{der} U(X, 1) - F(L(X)) \cdot \text{der} L(X, 1) + \int (L(X), U(X), \text{der} F(X, T, \partial X(X), \partial X(T)), T) \]  

Here \( L, U \), and \( F \) may be built-in, library, or user-defined functions, and \( \text{der} L, \) etc., represent their derivatives. We have used user-defined functions here to illustrate the most general case; built-in functions yield simpler results because their derivatives (\( \text{der} F, \) etc.) are expressed in terms of other built-in functions.

For example,

\[ \int (0, X^2, (T \cdot X)^2, T) \]  
\[ \quad \partial X \]  
\[ \colct \]  
\[ \Leftrightarrow \int (0, X^2, 2 \cdot (\partial X(T) \cdot X + X(X) \cdot T) \cdot (T \cdot X), T) + 2 \cdot X^3 \cdot 2 \cdot X. \]

Although the integral in this result appears to be tractable, the presence of the \( \partial \)'s prevents it from being matched by any of the built-in patterns. Evaluating an integral expression does not evaluate the integrand (as you would expect for the arguments of most functions), so any further evaluation of this result does not remove the derivatives. To finish the evaluation, you have to extract the integrand, evaluate it (keeping \( T \) undefined), and then reapply the integration. Manually, you can extract the integrand by selecting it in the EquationWriter subexpression mode and using \( \text{ESUBE} \) to copy it to the stack, and \( \text{EREPL} \) to replace it after it is simplified. Alternatively, you can apply the program \( \text{INTEVAL} \), which evaluates an expression while ensuring that integrands within the expression get evaluated before the integrals.

<table>
<thead>
<tr>
<th>INTEVAL</th>
<th>Integrand Evaluation</th>
<th>CO9E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{level 1} )</td>
<td>( \text{level 1} )</td>
</tr>
<tr>
<td></td>
<td>'expression_1' ( \Leftrightarrow ) 'expression_2'</td>
<td></td>
</tr>
</tbody>
</table>

\( \langle \langle - \text{DUP} \rightarrow \text{fn} \rightarrow \text{fx} \rangle \rangle \)  
\( \langle \langle - \rightarrow \text{f1} \rightarrow \text{f2} \rightarrow \text{f3} \rightarrow \text{f4} \rangle \rangle \)  
\( \langle \langle \text{f1} \rightarrow \text{f2} \rightarrow \text{f3} \rightarrow \text{f4} \rightarrow \text{f} \rangle \rangle \)  
\( \Leftrightarrow \)  
\( 'fx' \rightarrow \text{STO} \rightarrow \text{fn} \)  
\( \{ \langle \text{f} \rangle \rightarrow \langle \text{f1} \rightarrow \text{f2} \rightarrow \text{f3} \rightarrow \text{f4} \rangle \} \)  
\( \langle \langle \text{MATCH} \rightarrow \text{DROP} \rightarrow \text{EVAL} \rangle \rangle \)  
\( \Leftrightarrow \)  
Create local variables.  
User-defined function definition.  
Store in \( \text{fx} \).  
Replacement pattern.  
Replace the integral.  
Evaluate the expression.
If you execute INTEVAL on the result above, you obtain

\[
\int (0, x^2, 2*\text{T}^2 + x^2) + 2*x^3 + 2*x.
\]

The derivatives are gone, but the integrand still does not match an allowed integral pattern. However, COLCT helps here, by transforming the $T^2$ into $T^2$:

\[
\text{COLCT EVAL EVAL} \quad '2*(x^2 + 3/3) + 2*(x^3 + 2)*x'.
\]

This result can be simplified further by using POLY (section 17.4.1):

\[
'x' \quad \text{POLY} \quad '2.66666666667*x^2'.
\]

The case of integration of a derivative is more straightforward than the opposite, so the HP 48 realization is also correspondingly simpler. The definition

\[
\int_{\text{u}}^{\text{l}} \frac{\partial}{\partial t} F(t) dt = F(u) - F(l)
\]

translates on the HP 48 to

\[
\text{\int_{(L,U,\text{T}(F(T)),T')} EVAL EVAL} \quad 'F(U) - F(L)'.
\]

18.4.1.3 Adding Integration Patterns

The list of integration patterns built into the HP 48 is fixed, and there is no provision for adding additional patterns for the \int command itself. However, it is possible to use MATCH commands to program a substitute for \int that tests integrals against an extensible list of patterns. The program XINT listed below demonstrates one such approach.

XINT uses a list of patterns stored in a variable IPATS (Integral PATternS). Each pattern is represented by a MATCH list, where the first object is an integrand pattern containing $t$ as the integration variable, and the second is the antiderivative. The third object is the optional MATCH test (section 17.3). For example, the MATCH patterns needed for

\[
\int_{a}^{b} \frac{1}{x(x+1)} dx = \ln \left( \frac{x}{x+1} \right) \bigg|_{x=a}^{x=b}
\]
are represented by the list

\[
\{ \frac{1}{t+1} \}' \ln(\frac{t+1}{t+1})'.
\]

With this list stored as one element of the IPATS list,

\[
\int(A,B,\frac{1}{x(x+1)})x' \text{ XINT EVAL } \not\equiv \ln(B/(B+1)) - \ln(A/(A+1))'
\]

A powerful application of the optional MATCH test is to permit the generalization of some patterns in a manner similar to that of \( f \) itself. For example, the above pattern can easily be generalized to integrands of the form \( \frac{c}{x(x+d)} \), where \( c \) and \( d \) are constants. However, the IPAT entry should include a test to ensure that \( c \) and \( d \) are not functions of \( x \):

\[
\{ \frac{1}{t+1} + 2 \}' \frac{1}{t+2}' \ln(\frac{t+1}{t+2})'.
\]

\[
\not(f(\text{QUOTE}(\text{QUOTE}(t+1),t+1) OR (\text{FUNOF}(\text{QUOTE}(t+1),t+1))))
\]

The test here uses a user-defined function \( \text{FUNOF?} \) (listed below), which uses \( \text{MATCH} \) to check its first argument for the presence of its second argument.

Another extension provides for cases where the integration variable appears uniformly as a linear form. The test for these cases is provided by the user-defined function \( \text{LINEAR?} \). It uses \( \text{OBJ-} \) to dissect its first argument (much in the manner of \( \text{RPN} \) in section 16.6), looking for instances of a name specified by the second argument. It returns \( \text{true} \) if the name appears at least once, and only to first-order.

Using \( \text{LINEAR?} \), the integration of \( \cos x \cdot \sin x \) can be represented in IPATS by

\[
\{ \cos(\text{QUOTE}(\text{QUOTE}(t+1),t+1)) \}
\]

\[
\not(\text{LINEAR}(\text{QUOTE}(\text{QUOTE}(t+1),t+1))))
\]

Then

\[
\int(A,B,\cos(C*x) \cdot \sin(C*x))x' \text{ XINT -2 SF EVAL EVAL } \not\equiv -(\cos(2*(C*B))/(2*C)) + \cos(2*(C*A))/(2*C)'
\]
<table>
<thead>
<tr>
<th>LINEAR?</th>
<th>Linear Form Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2</td>
<td>Level 1</td>
</tr>
<tr>
<td>'expression', 'name'</td>
<td>'flag'</td>
</tr>
</tbody>
</table>

```
<< e x
<< 0 0 1 0 0 - args lin go non occ
<< << 1 SWAP START lin EVAL NEXT >>
'args' STO
<< sube
<< IF go
THEN
CASE sube TYPE 9 SAME
THEN sube OBJ-
CASE OVER NOT
THEN DROP2 END
{ NEG + - * / }
SWAP POS DUP NOT
THEN DROP 1 - non
<< args EVAL >> END
DUP 4 <
THEN DROP args EVAL END
SWAP DROP 5 SAME
THEN 1 - non << lin EVAL >>
lin EVAL END
occ SWAP lin EVAL
occ DUP ROT - ROT lin EVAL
occ ROT -
IF AND
THEN 0 'go' STO END
END
END
sube x SAME
THEN 1 'occ' STO+
IF non
THEN 0 'go' STO END
END
END
>>
>> 'lin' STO
e lin EVAL
go occ AND
>>
>>
```

- Initialize local variables.
- Subroutine for multiple arguments
- Main test routing
- Continue if go is true.
- If current object is an expression, then take it apart.
- Do nothing for zero-arg functions
- If non-linear, then make non true and check the arguments.
- Handle +, -, NEG.
- For /, treat denominator as non-linear, and numerator as linear.
- For *, check first argument, and second arguments.
- If x appears in both, then quit.
- Zero arguments. Is it x?
  - Then increment the occurrence count.
  - If inside a non-linear expression, then quit.
- Save the subroutine.
- Test the original argument.
  - Return true if x was found, with no non-linear occurrences.
### FUNOF?

<table>
<thead>
<tr>
<th>level 2</th>
<th>level 1</th>
<th>flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>'expression'</td>
<td>'name'</td>
<td>0BEF</td>
</tr>
</tbody>
</table>

- `<< f x` - Store arguments.
- `<< f x DUP 2 -LIST \( \text{IF MATCH} \)` - Test if \( f \) contains \( x \).

#### XINT

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'expression(_1)'</td>
<td>'expression(_2)'</td>
</tr>
</tbody>
</table>

- `<< << - low up int x` - Start definition of \( f \).
- `<< int x \{ t\} + \text{MATCH DROP}` - Replace integration var. with \( t \).
- `IPATS DUP SIZE 0 \rightarrow \text{pats n t}` - Set up for loop through patterns.
- `<< \text{DO pats n GET \text{UNTIL MATCH} 't' \text{STO}}` - Get the next pattern.
- `'n' \text{DECR NOT t OR \text{END}}` - Repeat until a match, or list is exhausted.
- `IF \text{t THEN DUP up 2 -LIST 'f' APPLY \text{SWAP low 2 -LIST 'f' APPLY \{ 'f\&(1,2)\}' \&1\{t1=\&2\}}} \text{IMATCH DROP ELSE DROP low up int x} f \text{END}}` - If there was a match:
  - Dummy function of upper limit.
  - Dummy function of lower limit.
- `<< EVAL \{ 'f\&(1,2,3,4)'
  'f\&(1,2,QUOTE(3),QUOTE(4))'
  \} IF \text{IMATCH THEN EVAL END \text{END}}` - Replace \( f \) with | forms.
- `<< EVAL \text{Try evaluating the argument.}
  \text{IF } f \text{'s remain, Replace } f \text{ with } f,` - No match; return integral.
- `<< EVAL \text{Store the function } f` - and execute \( f \).
18.4.2 Numerical Integration

The HP 48's symbolic integration capability is limited to a small number of integrand patterns, but even the list of all known closed-form indefinite integrals is fairly short. In many cases, especially in a practical problem-solving context, a numerical integral is as useful or even better than a symbolic integral. The HP 48, like its HP calculator predecessors back to the HP 15C uses a Romberg numerical integration method (see, for example, Press, Flannery, Teukosky, and Vetterling, *Numerical Recipes*, Cambridge University Press, 1986). The integrand is sampled at non-uniform intervals, which helps prevent errors due to periodicity in the function. The integrand is generally not sampled at the integration limits, which is helpful when the limits occur at a singularity or other difficult point.

The numerical evaluation of \( \int \) produces a series of increasingly accurate estimates of the integral, derived from sampling intervals that are halved at each iteration. The process terminates when three successive estimates differ by amounts less than an error tolerance that you specify, or after a maximum of sixteen iterations have produced no apparent convergence (at this point the integrand has been evaluated 65535 times). The error tolerance is determined by the current real number display setting: \( n \text{ FIX} \) (or \( \text{SCI} \) or \( \text{ENG} \)) specifies an error tolerance \( \epsilon = 10^{-n} \). This in turn relates to the probable error in the numerical integral:

\[
\text{error} \leq \epsilon \int |f(x)| \, dx.
\]

The use of the display format to determine the integration accuracy eliminates the need for an another explicit argument for \( \int \) in addition to those needed for symbolic integration (this use of the display format is similar to that of \(-\int\)--see section 16.3.1). The syntax for the two forms of integration is therefore the same, so that you don’t have to choose one form or another in advance. For example, if executing \( \int \) in symbolic execution mode returns an expression that still contains an integral, you can go ahead and obtain its numerical value by executing \(-\int\) on the expression.

As a function, \( \int \) can only return one result to the stack. However, it is also helpful to obtain an estimate of the error of the calculation as well as the integral value itself. \( \int \) returns the error estimate as a real number stored in a variable \( \text{IERR} \), which is created in the current directory if it does not already exist. A successful integration yields a positive value for \( \text{IERR} \); if the integrand is essentially always positive or always negative, you should find that the value is approximately \( \epsilon \) times the integral. For example, with a display setting of 5 \( \text{FIX} \),

\[
\int(0,1,\text{EXP}(X),X)' \rightarrow \text{NUM} \rightarrow 1.71828182875,
\]

and then
IERR \[= 1.71814488996 \times 10^{-5}\]

(here we are showing the results in STD format). Dividing the latter result by the integral value yields \(9.9992 \times 10^{-6}\), which is comparable to the error tolerance \(10^{-5}\) derived from the display setting. When the integrand is a reasonably well-behaved function, the actual error can be substantially less than the prediction represented by IERR. In this example, the result differs from the correct value \((e-1)\) only in the tenth decimal place.

If the integration results have not converged after sixteen interations, the number \(-1\) is stored in IERR. The value returned to the stack is the last estimation of the integral.

The choice of error tolerance for any integral is a trade-off of numerical precision against execution time. Reducing the error tolerance increases the number of integrand samples that are necessary. Consider, for example, the following integral:

\[
\int_{0}^{\pi} (5+x \cos 3x) \, dx = 5\pi - 2/9 \approx 15.4857410458
\]

Evaluating the object \('\int (0,\pi,5+x\cos(3\times X)),X)'\) with different display formats, we find:

<table>
<thead>
<tr>
<th>Display</th>
<th>Result</th>
<th>Error</th>
<th>IERR</th>
<th>Samples</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 FIX</td>
<td>15.6</td>
<td>0.1</td>
<td>1.6</td>
<td>7</td>
<td>0.8</td>
</tr>
<tr>
<td>3 FIX</td>
<td>15.486</td>
<td>5.5E-6</td>
<td>0.015</td>
<td>31</td>
<td>2.8</td>
</tr>
<tr>
<td>5 FIX</td>
<td>15.48574</td>
<td>8.3E-8</td>
<td>1.5E-4</td>
<td>63</td>
<td>5.4</td>
</tr>
<tr>
<td>7 FIX</td>
<td>15.4857410</td>
<td>0</td>
<td>1.5E-6</td>
<td>127</td>
<td>10.7</td>
</tr>
<tr>
<td>9 FIX</td>
<td>15.48574104</td>
<td>1E-10</td>
<td>1.5E-8</td>
<td>127</td>
<td>10.8</td>
</tr>
<tr>
<td>STD</td>
<td>15.4857410457</td>
<td>1E-10</td>
<td>1.5E-10</td>
<td>255</td>
<td>21.3</td>
</tr>
</tbody>
</table>

Notice that reducing the error tolerance below \(\varepsilon = 10^{-7}\) (7 FIX) doesn't appreciably improve the integration, which has reached the intrinsic accuracy limit of the calculator. This is because the integrand is a reasonably smooth curve in the region of interest, and can be accurately represented by a fitted curve based on the 127 sample points.

It is interesting to show where an integrand is actually sampled, which you can do with the following program. INTSAMP takes an expression containing an integral, and replaces the integrand with a function that plots each value that it returns as it is integrated.

-609-
Using INTSAMP on the current example,

\[ 1 \text{ FIX } -.5 \ 3.5 \ \text{XRNG } -.5 \ 8 \ \text{YRNG} \ \int_{0}^{\pi,5} x \cos(3x) \, dx \]

INTSAMP yields this picture:

The seven sample points make a fairly sparse representation of the function. Repeating the exercise after 5 FIX plots 63 sample points:
You can see that increasing the number of sample points in this example should not make a dramatic improvement in the integration accuracy.

18.4.3 Integration Strategies
Not all integrands are nice smooth functions like $x \cos x$, and so will not always numerically integrate so nicely. For example, consider the integral

$$\int_{-0.5}^{2.5} \sqrt{|x-1|} \, dx$$

Plotting the integrand from $-0.625$ to $2.625$ (y-range $-0.4$ to $1.5$) shows a cusp at $x=1$:

The cusp makes the approximation of the function for numerical integration difficult; with an error tolerance of $10^{-5}$, executing

$$f(-0.5,2.5,\sqrt{ABS(X-1)},X) \quad \text{NUM} \quad 2.44948455085$$

takes nearly four minutes (4095 samples).

In this case, and in many other numerical integration problems, a simple change in the
setup of the problem can make a great deal of difference in the integration time and the accuracy of the result. One useful strategy is to break up the domain of the integration into two or more smaller regions. In the current example, the cusp at \( x = 1 \) is the obvious segmentation point:

\[
\int(-0.5, 1, \sqrt{\text{ABS}(x-1)}, x) + \int(1, 2.5, \sqrt{\text{ABS}(x-1)}, x)
\]

\[
\quad\rightarrow \quad \text{NUM} \quad 2.44948983536
\]

takes only two seconds, and the result is accurate to two more decimal places than the previous result.

The program SEGINT automates the process of segmenting an integral. It takes as arguments an expression containing an integral, and a second object that specifies the segmentation point. It returns a new expression containing the original integral split into two parts:

\[
\int(-0.5, 2.5, \sqrt{\text{ABS}(x-1)}, x) \quad \text{SEGINT} \quad \int(-0.5, 1, \sqrt{\text{ABS}(x-1)}, x) + \int(1, 2.5, \sqrt{\text{ABS}(x-1)}, x)
\]

SEGINT goes to some trouble to avoid any evaluation of the integrand or limits, so that you can check its result before proceeding with the integration. If you want to skip that step, you can use NSEGINT:

\[
\int(-0.5, 2.5, \sqrt{\text{ABS}(x-1)}, x) \quad \text{NSEGINT} \quad 2.44948983536
\]
Not all cases have segmentation points as obvious as that in the previous example. For example, consider the integral

\[ \int_{1}^{\infty} \frac{1}{x(x+1)} \, dx \]

If you attempt to evaluate this directly on the HP 48, you obtain a nonsense answer:

\[ \int (1, \text{MAXR}, 1/(X*(X+1)), X) ' \, \text{NUM} \rightarrow \text{0} \]

This is a numerical precision problem: the integrand is effectively zero over most of the domain between the limits. You can do better by using NSEGINT to divide the domain into two parts:

5 \text{ FIX } \int (1, \text{MAXR}, 1/(X*(X+1)), X) ' \, 100 \text{ NSEGINT \rightarrow 0.68320} \]

It is hard to have confidence in this result, since there is nothing special about the segmentation point \( X = 100 \). Indeed, using \( X = 1000 \) instead returns 0.69215, suggesting that these approximations may only be accurate to one or two places.

A better strategy is transform the integral into one more suited for numerical integration, by an appropriate change of variables. That is, if we define a variable \( y = g(x) \), then an integral

\[ \int_{x_1}^{x_u} f(x) \, dx \]

can be re-expressed as

\[ \int_{g(x_1)}^{g(x_u)} f(g^{-1}(y)) \cdot \frac{d}{dy}(g^{-1}(y)) \, dy \]
For integrals where one or both limits are infinity, a particularly useful transformation is
\( y = \tan(x) \), which maps the entire real \( x \)-axis onto the interval \( -\pi/2 \leq y \leq \pi/2 \).

The program CHVAR listed next transforms an integral expression according to the
rules given above, with the variable change specified by an equation of the form
\( y = g(x) \). For the current example, we will use the arc-tangent transformation:

\[
\int (1, \text{MAXR}, (1/(x*(x+1)), x)^' \quad Y = \tan(x) \quad \text{CHVAR}
\]

\[
\int (1, \tan(1), \text{MINR}, (1/(\tan(y)*(-\tan(y)+1))*(-\tan(y)+1)^2), y)^'
\]

This symbolic result is not particularly illuminating, except that you can observe that the
integration limits have become \( \tan(1) = \pi/4 \) and \( \tan(\infty) = \pi/2 \). The result easily
integrates numerically:

\[
\text{STD} \quad \text{NUM} \quad .693147180555
\]

which compares well with the exact result \( (\ln 2) \), which is 0.693147180560 to twelve
places.

<table>
<thead>
<tr>
<th>CHVAR</th>
<th>Change of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{level 2 level 1} )</td>
<td>( \text{level 1} )</td>
</tr>
</tbody>
</table>
| \( 'f(x)dx' \quad 'y = g(x)' \) | \( 'f(h(y))dy' \)

\[
\ll \quad \text{DUP} \quad \text{EQ-} \quad 4 \quad \text{ROLL} \quad \text{OBJ-} \quad \text{DROP2}
\]

\[
\ll \quad \text{RCLF} \quad \text{-} \quad \text{y g low up f x flags}
\ll \quad \text{x ISOL} \quad \text{EQ-} \quad \text{SWAP} \quad \text{DROP} \quad \text{- ginv}
\ll \quad \text{g x low} \quad 2 \quad \text{- LIST} \quad \text{tMATCH} \quad \text{DROP}
\ll \quad \text{g x up} \quad 2 \quad \text{- LIST} \quad \text{tMATCH} \quad \text{DROP}
\ll \quad \text{ginv y FDER} \quad \text{*}
\ll \quad \text{y 4} \quad \text{- LIST} \quad \text{t} \quad \text{APPLY}
\ll \quad \{ \quad 't+([1,2,3,4])' \quad \text{f([1,2,3,4])} \quad \}
\ll \quad \text{tMATCH} \quad \text{DROP}
\ll \quad \text{flags STOF}
\ll \quad \gg
\ll \quad \gg
\ll \quad \gg

Note that unlike the other programs in this section, CHVAR expects its first argument to
be an algebraic object containing only an integral. CHVAR uses the program FDER (section 18.1.2) to compute unevaluated derivatives.

18.4.4 Programs as Integrands
In the HP 28 and earlier calculators, it is possible to obtain a numerical integral by representing the integrand with a program (called *implicit* integration on the HP 28). The program contains no explicit variable of integration, but instead is designed to take a value from the stack (the integration sample point), and return the integrand value at that point.

Although the HP 48 does not specifically provide for program integration, it is simple to convert a program to an integrable form. It is only necessary to convert it to a user-defined function, then integrate that function. That is, given a program \( \langle \langle \text{program} \rangle \rangle \), you must recast it as

\[ \langle \langle \rightarrow x \langle x \text{program} \rangle \rangle \rangle \]

and store it in a variable. Then you can integrate it, e.g.

\[ \int (A,B,F(X),X) \rightarrow \text{NUM}, \]

where \( F \) is taken here as the user-defined function name.

As usual, it is possible to automate this process by means of a program:

<table>
<thead>
<tr>
<th>PRGINT</th>
<th>Program to Integral</th>
<th>8E67</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle \langle \text{program} \rangle \rangle )</td>
<td>lower</td>
<td>upper</td>
</tr>
</tbody>
</table>

\[ \langle \langle \rightarrow \text{low up prog} \langle \langle \rightarrow x \langle x \text{prog} \text{EVAL} \rangle \rangle \rangle \rangle \rightarrow \text{f} \]

\[ \langle \langle \int (\text{low,up,f(x),x}) \rightarrow \text{NUM} \rangle \rangle \]

Save arguments.
user-defined function.
Store as \( f \).
Evaluate the integral.
PRGINT takes three arguments: a lower limit, an upper limit, and a program for the integrand. The program should take one value from the stack, and return one value. PRGINT returns a numerical integral, computed to a precision determined by the number display setting. For example, you can compute the integral

\[ \int_{0}^{5} x^2 \, dx \]

with

\[
3 \text{ FIX } 0 \ 5 \ "\text{SQ}>" \ \text{PRGINT} \ \frac{41.667}{\text{RREF}}
\]

18.4.5 Multiple Integrals
Because \( \int \) is an HP48 function, it is straightforward to use it to compute a double integral, where the integrand itself contains an integral. Multiple integrals, where integrals are nested any number of times, present no additional difficulties.

**Example.** Find the area of a region bounded by the parabolas \( y = x^2 \) and \( y = 4 - x^2 \).

To help you visualize the problem, start by plotting the two curves:

\[
'\text{PPAR}' \ \text{PURGE} \ -12.4 \ 12.8 \ \text{YRNG} \ 'x^2=4-x^2' \ \text{STEQ} \ \text{ERASE} \ \text{DRAW}
\]

To find the points where the curves intersect, execute

\[
\text{RCEQ} \ 'x' \ \text{QUAD} \ \text{COLCT} \ \frac{\text{RREF}}{'}x=1.41421356237*\text{s1}' \text{,}
\]

which shows that the intersections are at \( x = \pm \sqrt{2} \). The area bounded by the two curves is therefore given by the integral
As an algebraic object, this integral is

\[ \int_{-\sqrt{2}}^{\sqrt{2}} \int_{x^2}^{4-x^2} \frac{1}{y} \, dy \, dx \]

which looks like this in the EquationWriter:

\[ \int_{-\sqrt{2}}^{\sqrt{2}} \int_{x^2}^{4-x^2} 1 \, dy \, dx \]

Executing EVAL once, you obtain:

\[ \int_{-1.41421356237}^{1.4142135627} (4-x^2) - (x^2) \, dx \]

Because the integrand is invisibly quoted (section 18.4.1.2), further EVAL’s don’t change this result. COLCT helps, by resolving the where forms:

COLCT  \(\int_{-1.41421356237}^{1.4142135627} (4-x^2-x^2) \, dx \)

Then

EVAL EVAL  \(7.5424723266\)

This same result can be obtained in a single operation by applying \(-\text{NUM}\) to the original integral, but the double numerical integration takes longer than the symbolic evaluation to obtain the same accuracy.

18.4.6 Polynomial Approximations
The HP 28 provides an approximate form of symbolic integration, in which an integrand is converted to a Taylor’s polynomial which is then integrated. This primitive type of integration may not seem particularly useful, since an integral computed this way will
not much resemble the correct symbolic form of the integral unless the integrand happens to be a polynomial in the integration variable. But in many circumstances, you may be less interested in the precise mathematical form of an expression than in determining a reasonable view of its behavior over some region of interest. For this purpose, the HP 28 approximate symbolic integral is an improvement over numerical integrals, and you may want to apply the method on the HP 48 for integrands that do not match any of the built-in patterns. It is much faster to compute the value of a polynomial as you vary some parameter in the expression than to evaluate an integral numerically for the same values of the parameter.

As an example of the appropriate use of such an approximation, consider the solution of the equation

\[ \frac{1}{\pi} \int_{-\pi/2}^{x} t \cos^2(t) dt = x^3 - 4x - 2. \]

or, in HP 48 syntax,

'1/\pi * \int(- \pi/2, X, T*COS(T), T)=X^3-4X-2'

Even executing 3 FIX to reduce the numerical integration times, it takes over six minutes to plot this equation (default plot parameters, radians mode):

Then, using FCN ISECT with the cursor right at the rightmost intersection of the curves, it takes an additional 30 seconds to find the root of the equation at X = 2.203.

An alternate approach is to approximate the integrand on the left side of the equation with a polynomial:

'T*COS(T)' 'T' 5 TAYLR \(\equiv\) 'T-3/3!*T^3+5/5!*T^5'.

-618-
Then, integrate this result from $-\pi/2$ to $X$:

\[
'X' \text{ PURGE } -2 \text{ SF } '-\pi/2' \ 'X' \text{ ROT } 'T' \int \text{ EVAL COLCT
}\]

\[
\Rightarrow \ ' -0.57709430249 + 0.5X^2 - 0.125X^4 + 6.944444444455E - 3X^6'
\]

(Here we have set flag $-2$ to convert $\pi$ to a number, only to keep the final result short enough to be reasonably legible.) To create the approximate equation representing the original problem, divide this result by $\pi$, and equate it to 'X^3 - 4X - 2':

\[
\pi / 'X^3 - 4X - 2' = \text{STEQ}
\]

For this equation, DRAW takes only 37 seconds:

![Graph of X*COS(X) and its polynomial approximation]

ISECT (5 seconds) returns 2.205, which differs from the previous result by only 0.1%. For more accuracy, you can increase the order of the Taylor's polynomial. You can get an idea of the validity of the approximation by plotting 'X*COS(X)' and its polynomial approximation together:

\[
\{ 'X*COS(X)' 'X - 3/3!*X^3 + 5/5!*X^5' \} \text{ STEQ DRAW}
\]

![Graph of polynomial approximation]
The two curves are substantially the same in the range $|x| < 2$, and diverge elsewhere. Repeating the exercise with a higher order approximation yields a better match over the region of interest $-\pi/2 < x < 2.3$; for example, with a 9th order polynomial, the approximate root differs by 0.0003% from that obtained from the numerically evaluated integral.
19. The Time System

The HP 48 time system provides these general capabilities:

- An optional real-time “ticking” digital clock display, including the current date.
- Commands that return the current time and date.
- Time and date arithmetic.
- Appointment and object execution alarms.

Ordinary manual time operations—setting or adjusting the clock, setting alarms, and the alarm catalog—are straightforward and simple to use. This chapter will focus on aspects of the time system that are not emphasized in the Owner’s Manual, especially the computational and programmable aspects of the time system.

19.1 The Clock

The HP 48 clock is based on a quartz crystal that nominally oscillates at 32,768 Hz. The crystal is comparable in accuracy to that found in common quartz watches, except that it does not usually enjoy the same temperature regulation as a wristwatch that spends the day next to your skin, and so may not achieve a time accuracy quite as high. There is no means on the HP 48 to adjust the clock rate. (There is a program available on the Hewlett-Packard calculator electronic bulletin board that can determine the actual clock rate experimentally, and set periodic alarms to adjust the time to maintain better time precision.)

You can view the current HP 48 time by activating the “ticking” clock display in the status area. This happens automatically whenever you activate the alarm catalog (→TIME) or the time menu (TIME)

```
{ HOME } 05/01/92 09:00:18A
4:
3:
2:
1:
SET ADJ ALRM ACK ACKA CAT
```

“Ticking” refers to the second-by-second update of the time display. By default, the
The current time is encoded in HP 48 memory by means of a 52-bit RAM register and a 32-bit electronic counter. The register stores the scheduled time of the "next event"--the next time-dependent action that the HP 48 must take, such as the next alarm to come due, the next one-second display clock update, or the ten-minute inactivity calculator turn-off. (There is a second time counter that is used for cursor blink, which is therefore independent of the rest of the time system.) The counter is a countdown timer that represents the time remaining until that next event. The current "real" time then is the difference between the values in the next event register and the counter. That time can be anywhere in the range January 1, 1990 to Dec 30, 2089.

The counter decrements by one for each four crystal oscillations, meaning that the fundamental clock "tick" is actually 1/8192 second. With 32 bits for the counter, the longest the HP 48 can go without a time event is $2^{31}$ ticks, slightly more than 3 days. That occurs when the HP 48 is turned off, and no alarms are scheduled within that time--then the calculator wakes itself up just long enough to reschedule the next event time. Usually, all of this processing happens so fast that it is not noticeable. The only exception is when you have a large number of alarms stored in the alarm catalog (section 19.2.5); the time required to scan through the list of alarms to find the next due alarm could make the digital clock display miss an occasional one-second display update.

The time encoded by the next event register and the timer is deliberately designed to be protected from accidental reset. Even system halts and memory resets (section 5.8) do not affect the time data, even if a reset is caused automatically by the detection of a memory fault. A special checksum is maintained for the next event register; only if that checksum indicates a problem with the time register content itself is the clock reset to the default of midnight, January 1, 1990.

19.1.1 Setting and Reading the Time

The HP 48 uses ordinary real numbers to represent the time and date in stack operations. A time number has the format \( hh.mmssssssss \), where \( hh \) is one or two digits representing the hour 0-23, and \( mm \) are two digits representing the minute 0-59. The first two digits \( ss \) are a whole number 0-59 of seconds; the remaining digits encode the decimal fraction seconds. Thus 11.0519801 represents 11 hours, 5 minutes, and 19.801 seconds.

Date numbers are real numbers used to represent calendar dates. These numbers use
one of two forms mm.ddyyyy, or dd.mmyyyy, depending on whether the the date format flag –42 is clear or set, respectively. Here the digits dd are the day number 1-31, mm is the month 1-12, and yyyy is the four-digit year. You can switch the format by setting or clearing flag –42, or by pressing M/D.

The time-set menu (\[ TIME \] SET\) provides menu keys for setting the clock manually:

- \[ \Rightarrow \] TIME\ takes a time number from the stack and sets the clock to that time.
- \[ \Rightarrow \] DATE\ uses a date number to set the system date.
- \[ A/PM\] switches the time between a.m. and p.m. It is normally used after \[ \Rightarrow \] TIME\ when a post-noon time is entered in a 12-hour form, i.e. 3 \[ \Rightarrow \] TIME\ \[ A/PM\] sets the time to 3:00 p.m.

You can adjust the time by hour, minute, or second increments or decrements by using the time-adjust menu (\[ TIME \] ADJST\):

```
{ HOME } 05/01/92 09:00:27A
4:
3:
2:
1:
HR+ HR- MIN+ MIN- SEC+ SEC-
```

Each menu key adds or subtracts one of the indicated time units from the system time. \[ HR+\] and \[ HR-\] are handy when you change time-zones; the remaining keys are used to correct the time to match an external clock. Note that none of these keys affects the clock rate.

For programming purposes, the command TIME (in the second page of the \[ TIME \] menu) reads the HP48 clock and returns the current time in time number format. The inverse operation is \~ TIME (first page of the \[ TIME \] SET\ menu), which takes a time number from level 1, and sets the clock to match the specified time. Neither TIME nor \~ TIME is affected by flag –41—both work with time numbers expressed in 24-hour format.

For timing applications where speed and consistency is important for reading the clock, you can use TICKS (\[ TIME \] menu, page 2). This command returns the time as a
19.1 The Time System

binary integer, which represents the system clock time in ticks—units of 1/8192 seconds. The absolute magnitude of the number is not particularly useful; typically TICKS is useful for computing the time difference between two executions. An example is provided by the program TIMED, in section 12.9, which uses TICKS to mark the start and end of a timed execution. After the execution, when speed is not critical, the difference in the two times is converted to ordinary seconds.

A counterpart to TICKS is CLKADJ (CLocK ADJust), which adds a specified (real) number of ticks to the system time. CLKADJ, found in the second page of the MENU ADJ menu, is the programmable version of the clock-adjust operations in the first page. For example, to advance the clock by one hour, execute

8192 3600 * CLKADJ.

CLKADJ's argument can be number of ticks up to ±25,834,291,200, which is ±365 days (8192-3600-24-365). CLKADJ automatically corrects the date, if necessary. For this reason, using CLKADJ to shift the system time is preferable to using TIME and -TIME, e.g.

TIME 1 + -TIME.

The latter method is less accurate because it doesn't account for the time to execute TIME 1 +, and takes no account of possible date changes.

DATE (MENU TIME menu, second page) and -DATE (MENU TIME SET menu) read and set the date using date numbers, much like TIME and -TIME use time numbers. However, the date commands are sensitive to the date format flag; for example, if the date is July 1, 1992, then DATE returns 7.011992 when flag -42 is clear, or 1.071992 when flag -42 is set.

For program display purposes, you can obtain the combined time and date as a string object by using TSTR. This command takes a date number (level 2) and a time number (level 1), and returns a string like those used in the alarm set menu (MENU TIME ALARM):

-41 CF -42 CF 10.281991 18.12428 TSTR "MON 10/28/91 06:12:42P"

TSTR respects flag -41 (12/24 hour mode) and -42 (date format). It truncates its time argument to integer seconds. A simple application of TSTR is shown by the following program, which takes a date number and returns a three-letter string for the day of the week to level 2, and a number 1-7 for the numerical day of the week (Sunday=1).
19.1.2 Time Arithmetic

The third page of the [TIME] menu contains four commands that assist with time arithmetic, by providing for conversion between time numbers and their equivalents in decimal hours, and for direct addition and subtraction of time numbers. A more thorough implementation of time (or date) arithmetic on the HP 48 would have provided separate object types for these quantities, so that ordinary + and – could be used. That approach would, however, have consumed considerable additional built-in ROM, at the expense of other HP 48 features. The use of time numbers in the hh.mmmsss format at least has historical precedent in HP calculators, all the way back to the HP 45.

Direct addition and subtraction of time numbers is provided by the commands HMS+ and HMS–. Each takes a pair of time numbers, and returns the sum or difference as a time number. Thus, for example, to determine the time that is 3 hours and 45 minutes later than 8:25:37 AM:

8.2537 3.45 + HMS+ = 12.1037.

These commands do not constrain their results to the normal time number range 0-24 hours--their action is to convert the minutes/seconds parts of the time numbers to decimal hours, perform the addition or subtraction, then convert the decimal part of the result into .mmssss format. This means that you can apply the operations equally well to angles measured in degrees, minutes and seconds:

45.12 56.37 HMS– = -11.25

finds the angle that is 56°37' less than 45°12'.

→HMS converts an angle or time from ordinary decimal form into the time number format:

1 R→D →HMS = 57.1744806247
shows that 1 radian is $57^\circ 17' 44'' .806247$. Following the usual HP 48 naming convention, the inverse of "HMS" is HMS-.

### 19.1.3 Date Arithmetic

The date arithmetic analog of HMS+ is DATE+, which computes a new date by adding a number of days (level 1) to an initial date (level 2). For example, the one hundredth day of 1992 is found by

```
-42 SF 31.121991 100 DATE+ □ 9.041992,
```

which is 9 April, 1992. In the example, we used the dd.mmyyyy format for the date numbers; as you might expect, DATE+ is sensitive to the date format flag -42. The number of days argument can be positive or negative.

You can also determine the number of days between two dates by executing DDAYS with the two date numbers as its arguments. For example, to find the number of days between July 1, 1975 and February 17, 1978:

```
-42 CF 7.011975 2.171978 DDAYS □ 962
```

Notice that the earlier date is entered first to return a positive result.

The allowed range of dates for the date arithmetic commands is from October 15, 1582 to December 31, 9999, which spans a range of 3,074,323 days. The latter number is the maximum number-of-days argument permitted by DATE+. A date number argument outside of this range causes the Invalid Date error.

The next two programs provide a combined form of date and time arithmetic. The first, ADD&T, adds a specified number of days and hours (expressed as a time number with minutes and seconds) to an initial date and time, returning a final date number and time number:

```
1.151991 12.30 32 4.15 ADD&T □ 2.161991 16.45
```

(with flag -42 clear) adds 32 days, 4 hours and 15 minutes to January 15, 1991, 12:30 p.m., returning February 16, 1991 at 4:45 p.m.

The second program, DD&T, finds the difference in days and hours between two events each specified by a date number and a time number:

```
2.161991 16.45 1.151991 12.30 DD&T □ -32 -4.15
```

-626-
shows that January 15, 1991 is 32 days, 4 hours and 15 minutes earlier than February 16, 1991.

<table>
<thead>
<tr>
<th>ADD&amp; T</th>
<th>Add Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 4</td>
<td>level 3</td>
</tr>
<tr>
<td>$date_1$</td>
<td>$time_1$</td>
</tr>
</tbody>
</table>

$\llparenthesis$ SWAP 24 * HMS+
HMS+
DUP 24 / FLOOR
ROT OVER DATE+
SWAP 24 * ROT SWAP HMS-
$\rrparenthesis$

$\Delta time$ in hours.
New time.
No. of days in new time.
Add to date.
Subtract from time.

<table>
<thead>
<tr>
<th>DD&amp; T</th>
<th>Delta Days and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 4</td>
<td>level 3</td>
</tr>
<tr>
<td>$date_1$</td>
<td>$time_1$</td>
</tr>
</tbody>
</table>

$\llparenthesis$ 4 ROLL ROT DDAYS 24 *
SWAP ROT HMS- HMS+
DUP 24 / IP
SWAP OVER 24 * $ightarrow$
$\rrparenthesis$

Initial $\Delta days$ in hours.
Total time difference.
Actual $\Delta days$.
Remaining $\Delta time$.

19.2 The Alarm System

An HP 48 alarm is an event that is scheduled to occur at a specified date and time. There are two types of alarms, according to the type of action associated with the scheduled events:

- An appointment alarm sounds an audible signal and displays a message.
- A control alarm executes an object, usually a program or the name of a program.

The HP 48 can store any number of both kinds of alarms (limited by available memory). The alarms are stored as a list in a global variable in a normally invisible subdirectory of the home directory (see section 5.3.2). The alarm list is not accessible through ordinary variable commands like STO or RCL because the data in the list is represented by uneditable system objects for fast access by the time system. You can view the contents
of the alarm list by activating the alarm catalog (section 19.2.5), or by recalling individual alarms with RCLALARM (section 19.2.4).

When the clock reaches a time that matches an alarm time, we say that the alarm *comes due*. Alarms that have not yet come due are called *pending alarms*; alarms set for times earlier than the current time are called *past alarms*. The calculator also keeps track of *past due* alarms--alarms that have come due but have not been acknowledged (see below).

An alarm is specified by four parameters:

- The *date* of the alarm.
- The *time*.
- The *repeat interval*, for alarms that automatically reschedule themselves at constant intervals. This parameter is zero for single-occurrence alarms.
- The *alarm execution object*. For appointment alarms, this object is a string object; for control alarms, it may be any other type of object, which is executed when the alarm comes due.

### 19.2.1 Setting Alarms Manually

Setting an alarm consists of storing the four alarm parameters into the alarm list. For manual entry, the best way to do this is to use the *alarm-set menu* (HOME TIME ALRM):

When you first enter the menu, this display looks as above. As you enter alarm parameters, the display in the stack area changes to reflect the new entries (the initial default is the current date at 12 a.m.). For example, entering 9.3 >TIME stores 9:30 a.m. as the alarm time:
Enter alarm, press SET
FRI 05/01/92 09:30:00AM

Enter alarm, press SET
FRI 05/01/92 09:30:00AM

$\textbf{DATE}$, $\textbf{TIME}$, and $\textbf{A/PM}$ work the same way as they do in the time-set menu (section 19.1.1). $\textbf{RPT}$ activates another menu for entering repeat intervals:

Select repeat interval

<table>
<thead>
<tr>
<th>1:</th>
<th>2:</th>
<th>3:</th>
<th>4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEK</td>
<td>DAY</td>
<td>HOUR</td>
<td>MIN</td>
</tr>
</tbody>
</table>

Here you enter a number of time units, using any of the units shown in the menu labels, then press the corresponding menu key. For example, for a weekly meeting alarm you would enter 1 $\textbf{WEEK}$ (7 $\textbf{DAY}$ would work as well). If you change your mind and don't want to enter a repeat interval, or you want to clear a previously entered repeat interval, press $\textbf{NONE}$ . After pressing any of the repeat menu keys, the initial alarm-set menu is restored.

$\textbf{EXEC}$ stores the alarm execution object, which also determines whether the alarm is an appointment alarm or a control alarm. Entering a string object creates an appointment alarm; any other object creates a control alarm. There is no special provision here for entering a string—you must use $\rightarrow$ " " and $\alpha$ as usual to enter a string. If you want to modify a string already entered (such as when you are editing an alarm using $\textbf{EDIT}$ in the alarm catalog), $\rightarrow$ $\textbf{EXEC}$ recalls the current string to the stack, where you can use $\uparrow$ or $\downarrow$ $\textbf{EDIT}$ to change it.

Of the four alarm parameters, only the date and time are strictly required. If you do not enter a repeat interval, zero is taken as the default, which means a non-repeating alarm. If you don't supply an execution object with $\textbf{EXEC}$ , the alarm will be an
appointment alarm that only displays the date and time when it comes due.

You can enter the alarm parameters in any order, and change the settings as many times as you want. The alarm is not actually stored in the alarm list until you press $\text{SET}$. When you do so, the alarm is set, and the menu is automatically changed to the main time menu.

Between the time when you first enter a parameter in the alarm-set menu and when you press $\text{SET}$, the pending alarm parameters are stored as a list in a global variable ALRMDAT created in the current directory. The list has the format used by STOALARM (section 19.2.4). Storing the pending alarm data this way during alarm entry allows you to maintain access to other calculator resources—in particular, you can change menus and then return to the alarm-set menu without losing the entry in progress. If the alarm-set menu is activated when ALRMDAT already exists, the display will show the parameters that are stored there. When you finally press $\text{SET}$, ALRMDAT is automatically purged. (Changing directories during alarm entry forces the start of a new alarm. The ALRMDAT in the original directory will remain in place until you either purge it, or return to the directory and complete entering that alarm.)

19.2.1.1 Cancelling Alarm Repeats
An alarm interval that is too small may cause an endless loop when it comes due, because it executes and reschedules too fast for you to get a chance to delete it. Even a system halt (section 5.8) won't help, because that (by design) does not affect the rescheduling of alarms. You can escape from this predicament by pressing $\text{ON} - 4$ (together). This disables rescheduling of the next alarm, which allows you to activate the alarm catalog and change the repeat interval for the alarm, or to delete it altogether. Turning the calculator off, then on, re-enables rescheduling in case you used $\text{ON} - 4$ unnecessarily.

19.2.2 Appointment Alarms
An appointment alarm is an alarm defined with a string stored as its execution object. This type of alarm is intended for use as an appointment reminder: when it comes due, it is announced—the HP48 beeps, turns on the busy and the alert annunciators, and displays the day, date, time, and a one-line message. The “beep” in this case is a two-stage audible alarm: first a brief preliminary “chirp,” then, after about four seconds, a repeated two-tone “cuckoo” sound. The chirp gives you a chance to shut off the alarm (by pressing any key) before the more annoying cuckoo starts, which is helpful in a meeting or a classroom when you want to minimize the disturbance to other people. You can also prevent the audible alarm entirely by setting flag $-57$. (This does not affect other kinds of beeps—nor does setting flag $-56$, which suppresses error beeps and BEEP sounds, affect alarms). Of course, if you disable the audible alarm, you must be
watching the display to see an appointment alarm announced.

When the alarm chirps, its message is displayed in the medium font in the topmost display row. The alarm's day, date, and time are displayed in the second row:

```
(8) X
Important Meeting
FRI 05/01/92 09:30:00A
4:
3:
2:
1:
```

The cuckoo alarm sound repeats up to ten times, lasting until about fifteen seconds after the original chirp. During that time, you may acknowledge the alarm by pressing any key. This stops the alarm sound, turns off the busy and alert annunciators, and restores the previous display contents. If you fail to acknowledge the alarm with a key press, the alarm display and sound vanish, but the alert annunciator remains on. There are other causes that activate this annunciator; when you see it, the best thing to do is turn the HP48 off, then on, whereupon it will display a warning message that explains the alert. With an unacknowledged alarm, you will see:

```
(8) X
Warning: Alarm
4:
3:
2:
1:
```

This warns you to press \([\text{[TIME]}\) :
This display shows the oldest unacknowledged alarm. You can then go ahead and acknowledge the alarm by pressing \texttt{ACK}. If there are additional unacknowledged alarms, the next oldest is displayed, which you can then acknowledge, and so on. You can also execute \texttt{ACKALL (ACKA)} to acknowledge all past due alarms at once.

An appointment alarm announcement can occur at any time when the HP 48 is ready for key presses, i.e. whenever the busy annunciator is off. This means that an announcement can occur when any built-in environment is active, including the plot environment, the Matrix Writer, and the Equation Writer, but not when a command or a user program is executing. If an alarm comes due when the busy annunciator is on, the alert annunciator is turned on, but the alarm is not otherwise announced audibly or visibly until the current execution is complete.

The alarm-set menu always requires you to enter the date of an alarm as a date number. Sometimes, however, it is more convenient to enter days in terms of days of the week or other intervals, such as “one week from tomorrow” or “the Tuesday after next.” The program \texttt{NEXTDAY} listed below provides a means for doing this. You can use it to enter an alarm date to begin an alarm entry; it always exits to the alarm-set menu, where you can complete the entry. You can also execute \texttt{NEXTDAY} after starting an alarm, but you should take care not to change directories and thereby lose track of the data already stored in \texttt{ALRMDAT}. When you execute \texttt{NEXTDAY}, it displays this menu:
The first four menu keys add a predetermined number of days to the current alarm date:

- TMRW sets the alarm date to the next day—tomorrow.
- TM+1 sets the alarm date to two days later—the day after tomorrow.
- WEEK increments the alarm date by one week.
- MNTH increments the alarm date by four weeks.
- QUIT (in the second page) exits to the alarm-set menu without changing the current date.

NEXT sets up a new menu:

![Menu](image)

Here you press one of the seven menu keys (SAT is in the second page) to indicate the day of the week. Thus, to set an alarm for next Wednesday, you execute NEXTDAY, then press NEXT followed by WED.

2ND works the same way as NEXT except that it sets the alarm date to the specified day one week later than the next occurrence of that day. That is, to set an alarm for the Thursday after next, execute NEXTDAY, then press 2ND THU.

When NEXTDAY exits to the alarm-set menu, the current alarm data is not automatically displayed (that only happens when you activate the menu with ALRM, and when you press any of the menu keys). The data is displayed when you enter the next alarm parameter, or you can display it immediately by pressing REVIEW.
The Time System

19.2

NEXTDAY

Next ...Day Alarm Utility

33B8

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Code" /></td>
<td>If ALRMDAT doesn't exist, then create default.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Get current alarm date.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Start of TMENU list. Next ...day.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Second ...day.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Tomorrow.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Day after tomorrow.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>1 week from now.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>4 weeks from now.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>No change.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Display the menu.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Sunday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Monday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Tuesday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Wednesday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Thursday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Friday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Saturday.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Show menu for ...day cases.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Days from now to alarm.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Add one week if necessary.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Add 7 for 2nd ...day case.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>New date.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Store the new date.</td>
</tr>
<tr>
<td><img src="image" alt="Code" /></td>
<td>Return to the alarm—set menu.</td>
</tr>
</tbody>
</table>

-634-
19.2.2.1 Unacknowledged Repeating Alarms
When a repeating appointment alarm is not acknowledged by a key press as it comes due, it is simply rescheduled to the next repeat time and does not become past due. If you would prefer to be notified when you have missed such an alarm, you can set flag -43. This prevents repeating appointment alarms from being rescheduled until they are acknowledged, either by a key press or by one of the acknowledgement commands. A past-due repeating alarm then also leaves the alert annunciator on, and issues the alarm warning when the HP 48 is turned on.

19.2.3 Control Alarms
A control alarm is characterized by an execution object that is not a string. When a control alarm comes due, no audible or visible signal is given; rather, the alarm index (section 19.2.4) is placed on the stack, and the execution object is executed. Control alarms allow you to schedule calculator operations to take place without any manual intervention. For example, on an HP 48SX with a RAM card in port 1 configured as independent memory, you might schedule an automatic memory archive into a port variable SAVE to occur every night at 2 a.m. This is done with an alarm set for 2 a.m., with a 1 day repeat interval, and the following execution object:

```
<< DROP :1:SAVE DUP PURGE ARCHIVE >>
```

The initial DROP discards the alarm index, which is not needed in this case. In general, a control alarm execution program can use the index to find the alarm itself in the alarm list. In particular, since control alarms are not automatically deleted from the alarm list after execution, a program might begin by executing DELALARM with the index to delete the alarm.

Like appointment alarms, control alarms do not execute when they come due unless the busy annunciator is off and the HP 48 is ready for key presses. But control alarms are more disruptive than appointment alarms, in that they automatically terminate special environments. In the command line, interactive stack, the catalogs, and the plot environment, a control alarm coming due exits from the environment (as if [ATTN] were pressed), then executes the alarm execution object. If the alarm comes due while the EquationWriter or the MatrixWriter is active, the object currently entered in the environment is saved into level 1 before the alarm index. (For the EquationWriter, the object is saved as a string if it is newly entered or has been modified.)

The program CHIMES provides another example of a program useful as an alarm execution object. It "chimes the hour" by sounding a number of tones equal to the current clock time (truncated to hours). An appropriate use is to make CHIMES the execution object for an alarm set to come due on the hour, with a repeat interval of one hour.
The program SETCHIMES in the next section does this automatically.

<table>
<thead>
<tr>
<th>CHIMES</th>
<th>Hour Chimes</th>
<th>E5D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;= 1 TIME IP 12 MOD IF DUP NOT THEN DROP 12 END START 500 .5 BEEP .3 WAIT NEXT )</td>
<td>Get the hour. 12 chimes for 12 o'clock.</td>
<td>Sound the chimes.</td>
</tr>
</tbody>
</table>

19.2.4 Alarm Commands
The four commands in the second page of the alarm-set menu provide complete programmable control over the HP 48 alarm system. To start with, the alarm setting features of the first page of the menu are combined into a single command, STOALARM. This command uses a single argument, a list that combines the four alarm parameters:

\[ \{ \text{date} \ \text{time} \ \text{execution-object} \ \text{repeat} \} \]

Here date and time are real numbers in the date-number and time-number format, respectively. execution-object is the alarm execution object, which may be any object—a string indicates an appointment alarm. The final argument repeat is an integer real number that specifies the alarm repeat interval, in ticks (section 19.1.1). A repeat interval of zero defines a non-repeating alarm. The maximum repeat interval is 999999999999 ticks, about 3.88 years.

When STOALARM executes, it adds an alarm to the current alarm list with parameters matching those in its list argument. For example, the following sets a weekly appointment alarm for 9:00 a.m., starting on February 2, 1992:

\[ \{ 2.121992 \ 9 \ "Important Meeting" \ 4954521600 \} \text{ STOALARM } \leftrightarrow \ 6 \]

The value 4954521600 for the repeat interval is obtained from \(1 \text{ week} \times 7 \ \text{days/week} \times 24 \ \text{hours/day} \times 3600 \ \text{seconds/hour} \times 8192 \ \text{ticks/second} = 4954521600 \ \text{ticks}.\)

The number 6 returned here by STOALARM is just an example, which will vary according to the current alarm list in your HP 48. The number is the alarm index of the new alarm. The index is a real number that indicates the position of the alarm in the alarm list, where an index of 1 corresponds to the chronologically earliest alarm (including date and time). You may not have a very frequent need for the index returned by STOALARM, but that number is otherwise hard to determine afterwards, and it is easy
to discard if you don't need it.

SETCHIMES is an example of the programmed use of STOALARM to create an alarm. The alarm is defined with the hour-chiming program CHIMES from the previous section in its execution object.

<table>
<thead>
<tr>
<th>SETCHIMES</th>
<th>Set Chime Alarm</th>
<th>0B2E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; DATE )</td>
<td>Today's date.</td>
<td></td>
</tr>
<tr>
<td>TIME IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF DUP 23 ==</td>
<td>If the next hour is midnight,</td>
<td></td>
</tr>
<tr>
<td>THEN 0 SWAP 1 DATE+ SWAP</td>
<td>then set for midnight next day.</td>
<td></td>
</tr>
<tr>
<td>ELSE 1 +</td>
<td>Otherwise set for next hour.</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt; DROP CHIMES &gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29491200</td>
<td>Alarm execution object.</td>
<td></td>
</tr>
<tr>
<td>4 LIST</td>
<td>Repeat interval one hour (in ticks).</td>
<td></td>
</tr>
<tr>
<td>STOALARM DROP</td>
<td>Combine alarm parameters into a list.</td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Set the alarm.</td>
<td></td>
</tr>
</tbody>
</table>

An alarm index is used as an argument by RCLALARM and DELALARM. RCLALARM returns a parameter list for the specified alarm, or returns the Nonexistent Alarm error if there is no such alarm. If you execute it after the STOALARM example above, with the same index, the original list is returned:

```
6 RCLALARM \(\vartriangleright\) \{ 2.121992 9 "Important Meeting" 4954521600 \}
```

Another example is given by the program DELCHIMES, which uses RCLALARM to find the hour-chime alarm in the alarm list.

DELCHIMES uses DELALARM to delete the hour-chime alarm once it has been found by RCLALARM. In general, DELALARM deletes the alarm specified by its argument, reporting the Nonexistent Alarm error if no such alarm exists. You can also use DELALARM to remove all alarms by using zero for the alarm index. The program DELOLDALARMS is more selective: it deletes only alarms set for times earlier than the current time.
The final alarm command is FINDALARM, which finds an alarm by its date and time. In particular, FINDALARM returns the index of the first alarm in the alarm list that is set for a time equal to or later than the time specified by the argument. The latter may be either a date number, or a list \{ date time \}. A date number used by itself is equivalent to using a list \{ date 0 \}—FINDALARM returns the index of the earliest alarm set for that date. Executing 0 FINDALARM returns the index of the oldest past-due alarm.

19.2.5 The Alarm Catalog
The HP 48 time system includes an alarm catalog, which is similar in operation to the equation catalog (section 14.2.3.2) and the statistics catalog (section 20.1). The alarm catalog...
The Time System 19.2

catalog lets you review, edit, copy, and delete the alarms currently stored in the alarm list. It is activated by the CAT key in the TIME menu, or by the TIME menu from any other menu:

![Catalog Example]

The catalog shown here is typical; the display you obtain naturally depends on the alarms currently stored in your HP 48.

When you first activate the catalog, a triangle selection pointer points to the next alarm due--the same one that is displayed in the TIME menu. The alarms are displayed in chronological order, with past alarms above the pointer, and pending alarms below. As in the equation and statistics catalogs, you can move the selection up and down using ▲ or ▼ --one alarm at a time with the unshifted keys, one “page” of five alarms with ▲ ▲ and ▼ ▼, and to the start and end of the catalog with ▶ ▲ and ▶ ▼. The selected alarm is the subject of the menu operations, with the exception of EXECSE. The latter toggles the catalog display between the initial date/time/execution object display as shown above, and a display that shows only the execution objects:

![Detailed Alarm Display]

The remaining menu operations work as follows:

- PURGE deletes the selected alarm from the alarm list.
• `EDIT` also deletes the selected alarm, but copies its parameters to ALRMDAT in the current directory, and activates the alarm-set menu (exiting from the catalog). This allows you to change any of the alarm parameters, then reset the alarm by pressing `SET`. It is important to remember that the alarm is actually deleted by `EDIT`, because even if you decide not to change the alarm, you must still press `SET` to reactivate the alarm.

• `STK` copies the alarm parameters to the stack in a list, in the format used by STOALARM and RCLALARM (section 19.2.4). The message Copied to Stack is briefly displayed in the top row.

• `VIEW` displays all of an alarm's parameters, using the same display that you see in the alarm-set menu. The expanded display persists for about two seconds, or as long as you hold the menu key down.

The only keys other than the menu keys that are active in the catalog environment are `ENTER`, which duplicates the action of `STK`, and `ATTN`, which terminates the catalog.

19.2.6 Automatic Alarm Deletion

By default, both appointment and control alarms are automatically deleted from the alarm catalog when they are acknowledged. For appointment alarms, acknowledgement is accomplished by a key press as the alarm is announced (section 19.2.2), or for past-due alarms, by ACK or ACKALL. For a control alarm, “acknowledgement” is just the execution of the alarm execution object (section 19.2.3). Alarm deletion conserves memory, and helps prevent execution delays associated with long alarm lists.

You may prefer, however, to prevent this automatic deletion by setting flag -44. The old alarms in the alarm list are a record of past alarm events that you may find useful. Furthermore:

• When an appointment alarm is announced, pressing a key to acknowledge it also clears the alarm display. If the alarm is deleted, you can't even use the alarm catalog to review the alarm message afterwards.

• For irregularly scheduled events, it is convenient to keep one copy of the alarm in the alarm list. Then to schedule the next alarm, you can use the alarm catalog `EDIT` to modify one or more of the parameters.

• Old control alarms may contain programs that have other uses (although it is better for sake of time system efficiency to store long programs in variables, and use the variables' names as the alarm execution objects instead of the programs themselves).
20. Statistics

In the realm of statistical computation, the HP 48 provides a capable but not exhaustive set of commands that deal with data entry, simple sample statistics, linear and pseudo-linear regressions, probability, and plotting. The last topic is covered in sections 15.9-15.11; in this chapter we will concentrate on non-graphical analysis tools.

20.1 Data Entry

The traditional HP calculator method of accumulating sample data is the \( \Sigma + \) command. In its original versions, \( \Sigma + \) takes a number from the stack and adds its value and its squared value to running totals of these values from a series of such entries. \( \Sigma + \) also increments a count of the entries. From the three stored values, the total, mean, and standard deviation of the entered data can be computed. Some calculators, such as the HP 41, enhance the action of \( \Sigma + \) to accumulate pairs of numbers, allowing calculation of cross-correlations and regressions along with the other statistics.

The HP 48 follows the general \( \Sigma + \) model of its predecessors, but generalizes the command to work with data points in any number of dimensions. Furthermore, \( \Sigma + \) saves the entered data rather than just the summary statistics, which allows for data editing, data transformation, and greater accuracy. The data is saved as an \( n \times m \) matrix, each row of which represents the coordinates of a data point in \( m \)-dimensional space. The number of rows \( n \) indicates the number of points that have been entered. Sample statistics are computed and linear regressions are performed on the columns of the data matrix.

The statistics analysis commands are collected in the statistics menu, activated by \( \mathbf{\Sigma-STAT} \). Activating this menu when no data has been stored produces this display:

![Statistics Menu Example](image)

Notice the deliberate similarity to the plot and HP Solve entry menus. The \( \mathbf{NEW} \),
\[ \text{EDIT} \], \[ \text{STO} \], and \[ \text{NEW} \] keys play the same roles in the statistics menu as \[ \text{EDEQ} \] and \[ \text{STEQ} \] in the plot and solve menus, except that the reserved name variable for statistics is named \( \Sigma \text{DAT} \), and \[ \text{EDIT} \] activates the matrix writer for editing. \( \Sigma \text{DAT} \) is called the current statistics matrix, by analogy with current equation \( \text{EQ} \) (section 14.2.3). Like \( \text{EQ} \), \( \Sigma \text{DAT} \) may contain either the object of interest, in this case a matrix, or an indirect reference to the object—the name of another variable containing a matrix. Statistical operations are always directed to the current statistics matrix in the current directory. You can select one of several matrices for these operations by storing its name in \( \Sigma \text{DAT} \), with either \( \text{STO} \) or \( \text{STOX} \) (which is a shortcut for \( \Sigma \text{DAT}' \) \( \text{STO} \)).

The statistics catalog (\[ \text{CAT} \]) works generally like the equation catalog (section 14.2.3.2), except that it displays variables containing matrices. The first three menu keys also differ from those in the equation catalog. Each stores the selected variable name in \( \Sigma \text{DAT} \), and:

- \[ \text{1-VAR} \] activates the second page of the statistics menu, for commands related to single-variable statistics.
- \[ \text{PLOT} \] activates the third page, for commands related to the discrete plot types.
- \[ \text{2-VAR} \] activates the fourth page, for two-variable statistics commands.

The basic action of \( \Sigma + \) is to take a real \( m \)-element vector from the stack, and append it as a new last row to an \( n \times m \) matrix stored or named in \( \Sigma \text{DAT} \). If the designated variable does not exist, it is created by \( \Sigma + \), by storing the vector there (as a \( 1 \times m \) matrix). For 1-dimensional data, you can use \( \Sigma + \) either with a single real number or with a one-element vector. \[ CL\Sigma \] purges \( \Sigma \text{DAT} \), to prepare for entry of a new set of data.

Once the matrix is established, subsequent entries with \( \Sigma + \) can either be vectors (of the right length), or separate real numbers taken from the lowest stack levels. For example, consider the 3-dimensional data points

\[
(11,12,20) \\
(12,23,23) \\
(15,27,24) \\
(14,31,25)
\]

To start entering this data, press \[ CL\Sigma \], then

\[
\begin{bmatrix}
11 & 12 & 20 \\
\end{bmatrix} \Sigma + .
\]
The display shows this entry as ΣDAT(1):

The incomplete ΣDAT(2) indicates that the calculator is ready for entry of a second data point. Now that ΣDAT has been established, entering a data point as a vector is no longer necessary. You can enter the coordinates as separate stack entries (maintaining the same order):

\[
\begin{align*}
12 & \ 23 & \ 23 & \ Σ+ \\
15 & \ 27 & \ 24 & \ Σ+ \\
14 & \ 31 & \ 25 & \ Σ+.
\end{align*}
\]

Note that \[ \rightarrow \text{LASTARG} \] always returns a vector, even when you have entered separate numbers:

\[
\rightarrow \text{LASTARG} \quad \rightarrow \quad [\ 14 \ 31 \ 25 \ ]
\]

Σ+ will also enter several data points together if they are combined into a matrix the same width as the current statistics matrix. For correction of a bad entry, Σ− (\[ \rightarrow \text{Σ-} \ \] ) reverses the action of Σ+, stripping the statistics matrix of its last row, and returning it to the stack as a vector:

\[
Σ− \quad \rightarrow \quad [\ 14 \ 31 \ 25 \ ]
\]

Repeated execution of Σ− returns each successive last row, allowing you to edit one or more of them. You can then restore the entries by executing Σ+ the same number of times.
Although \( \Sigma^+ \) is convenient for entering small data sets, using \( \Sigma^+ \) for each data point may not be the best method for larger sets. When you are using a program to enter data automatically, it is more efficient to combine all of the data into a matrix, then use \( \Sigma^+ \) or \( \text{STO}\Sigma \) to store it (one large movement of memory contents is more efficient than several smaller ones). For manual entry, you can use the MatrixWriter (section 6.6) to enter all of the data, taking advantage of the MatrixWriter's facilities for reviewing and correcting the data before storing it. After exiting the MatrixWriter via \( \text{ENTER} \), you can

- Execute \( \Sigma^+ \) to append the new data to the current statistics matrix.
- Execute \( \text{STO}\Sigma \) to replace (or create) the current statistics matrix.
- Use \( \text{STO} \) to store the matrix, then store its name in \( \Sigma\text{DAT} \).

The last option is combined into a single operation by the statistics menu operation \( \text{NEW} \), which takes a matrix from the stack and prompts you for a name:

```
\text{HOME}

Name the stat data, press \text{ENTER}
```

When you press \( \text{ENTER} \), the matrix is stored with the designated name, and the name is stored in \( \Sigma\text{DAT} \).

## 20.2 One-Variable Sample Statistics

The second page of the statistics menu (which is directly accessible with \( \text{PRG STAT} \)) contains five commands for computing simple one-variable sample statistics. (The sixth menu entry, \( \text{BINS} \) is associated with histogram plotting, and is described in section 15.11). For these commands, the columns in \( \Sigma\text{DAT} \) are treated as samples of independent variables, and the statistics of each sample are computed separately. For example, with the data set of the preceding section, we find

\[
\text{MEAN} \rightarrow [13 \ 23.5 \ 23]
\]

Each element of the result vector is the average of the numbers in the corresponding
column of \( \Sigma DAT \). The result object and those of the other commands in this section are single real numbers when \( \Sigma DAT \) contains only one column.

Similarly, you can use SDEV to compute the sample standard deviations of the data:

\[
SDEV \quad [1.82574185835 \quad 8.18026079454 \quad 2.16024689947]
\]

This standard deviation is defined by the formula

\[
\sigma_j = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2}
\]

where \( \sigma_j \) is the standard deviation for the \( j \)th column of data, \( x_{ij} \) is the \( i \)th entry in that column, and \( \bar{x}_j \) is the column mean. This definition constitutes an unbiased estimate of the standard deviation of the population from which the data sample was drawn. This differs from the standard deviation of the data itself. The latter is defined by the above formula with \( n \) in the denominator rather than \( n - 1 \), making it smaller than the sample standard deviation by a factor of \( \sqrt{n/(n-1)} \).

The HP 48 also provides the command VAR for computing sample variances, which are the squares of sample standard deviations. This command is not found in any menu, but you can execute it by name or via a custom menu.

MAXΣ and MINΣ compute the most positive and most negative values of the data in the separate columns:

\[
\text{MAXΣ} \quad [15 \quad 31 \quad 25]
\]

\[
\text{MINΣ} \quad [11 \quad 12 \quad 20]
\]

The midpoints of the data ranges are easily computed using these two commands:

\[
\frac{\text{MAXΣ} + \text{MINΣ}}{2} \quad [13 \quad 21.5 \quad 22.5].
\]

Because \( \Sigma DAT \) contains the actual entered data rather than summary statistics, it is straightforward to compute statistics other than those explicitly provided in the built-in command set. For example, the following program uses the sorting program SORT listed in section 11.5.3 to compute the medians of the data:
20.3 Two-Variable Statistics

The fourth page of the STAT menu contains commands related to two-variable statistics, which deal with the interrelationships between pairs of columns of data in ΣDAT. In many applications, the values in one column are considered to be samples of an independent variable, which may actually be a noise-free parameter rather than a random variable. The second column then represents a dependent variable, the values of which are measured or computed from the first variable. Alternatively, both columns may be treated as independent, where you are interested in the correlations between the pairs of data points.

In either case, you must designate which columns in ΣDAT represent the independent and dependent variables. This is accomplished by means of XCOL and YCOL, respectively (see also section 15.9), the names of which follow the HP 48 plotting convention of
associating the name \( x \) or \( X \) with the independent variable, and \( y \) or \( Y \) with the dependent variable. Each command takes a single real number, nominally a positive integer, and stores it as the independent or dependent column choice. The default choices are 1 for \( x \), and 2 for \( y \). These commands' menu keys are in the third page of the statistics menu. When that menu or the fourth page menu is activated, the status area shows the current choices:

\[
\begin{array}{c}
\text{xcol: 1} \\
\text{ycol: 2} \\
\text{Mod: LIN}
\end{array}
\]

The Mod: label at the right end indicates the current regression model; see section 20.3.2.

The current two-variable column selections are stored as real numbers in a list stored in the reserved-name variable \( Z\text{PAR} \). The list, which is analogous to \( P\text{PAR} \) (section 15.2.3), actually contains five entries:

\[
\{ \text{independent dependent intercept slope model} \}
\]

These objects specify, in order, the independent column number, the dependent column number, the regression intercept and slope coefficients (or their analogs depending on the regression model) from the most recent regression, and the name of the regression model. The first four objects are real numbers; the model object is the command that actually sets the model: \( \text{LINFIT EXPFIT, PWRFIT, or LOGFIT} \).

If \( Z\text{PAR} \) does not exist in the current directory when a command that uses it is executed, that command automatically creates \( Z\text{PAR} \) with the default values

\[
\{ 1 2 0 0 \text{ LINFIT} \}
\]

If the command is one that stores a value in \( Z\text{PAR} \), then that value is immediately substituted in the new list.
20.3.1 Correlations
The sample covariance of two sets of data is defined as
\[ \text{cov}(x,y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \]

where the \( x_i \) and \( y_i \) are the corresponding data points in the two sets, and \( \bar{x} \) and \( \bar{y} \) are their averages. The covariance is a measure of the mutual dependence of \( x \) and \( y \). It is positive if values \( x_i > \bar{x} \) tend to appear with values \( y_i > \bar{y} \) and \( x_i < \bar{x} \) with \( y_i < \bar{y} \), and negative if \( x_i < \bar{x} \) appears with \( y_i > \bar{y} \) and \( x_i > \bar{x} \) appears with \( y_i < \bar{y} \). A covariance of zero implies that the fluctuations of \( x \) and \( y \) around their averages are independent of each other.

The correlation coefficient \( \text{corr}(x,y) \) is a normalized version of the covariance:
\[ \text{corr}(x,y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}} \]

The coefficient has values of +1 for perfectly correlated variables, -1 for perfectly anticorrelated variables, and 0 for totally independent variables.

COV and CORR compute the covariance and correlation coefficient of the data in the two columns of \( \Sigma \text{DAT} \) specified in \( \Sigma \text{PAR} \). For example, consider two sets of 100 uncorrelated data points, created using \( \text{RAND} \) as follows:

```
.12345 RDZ 1 200 START RAND NEXT {100 2} -ARRY STO\( \Sigma \).
```

CORR should return a near-zero correlation coefficient:

```
1 XCOL 2 YCOL LINFIT CORR \( \equiv \) 4.85327453722E-2.
```

On the other hand, if one data set is derived from another by introducing a small amount of noise:

```
.12345 RDZ 1 100 START RAND DUP RAND .1 * + NEXT {100 2} -ARRY STO\( \Sigma \),
```

then CORR should return a coefficient close to one:
CORR \[ \cdot 9961417813403. \]

LINFIT is included in the sequence above because COV and CORR compute their results based on the ΣDAT data transformed according to the current regression model. See section 20.3.2.1.

20.3.2 Regressions

A linear regression is the computation of a linear relationship between two sets of data. If \( x \) and \( y \) denote the independent (XCOL) and dependent (YCOL) variables, then the linear relationship is \( y = ax + b \), and is referred to the regression of \( y \) on \( x \). In a graph of the line, \( a \) is slope, and \( b \) is the \( y \)-axis intercept. The regression used in the HP 48 is a least-squares fit, where values of \( a \) and \( b \) are computed such that the sum of the squared differences

\[
\sum_{i=1}^{n} (y_i - ax_i - b)^2
\]

is minimized. \( x_i \) and \( y_i \) are the individual measurements of the variables, and \( n \) is the number of pairs of measurements. In effect, the following simultaneous equations are solved:

\[
an + b \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i
\]

\[
a \sum_{i=1}^{n} x_i + b \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i.
\]

To obtain this type of regression, you can execute LINFIT to select the regression model (see the next section), then LR (Linear Regression) to compute the coefficients \( b \) and \( a \). The latter are returned to the stack as real numbers, with the tags Intercept: and Slope: to identify them. They are also stored as the third and fourth elements in the ΣPAR list.

To illustrate ordinary linear regression, we will first create some noisy “data:"

\[
.12345 \ RDZ \ 1 \ 100 \ FOR \ \times \ \times \ \DUP \ .5 \ \* \ 1 \ + \ \RAND \ + \ \NEXT \ \{ \ 100 \ 2 \ \} \ \-ARRY \ \STO\Sigma
\]

From this sequence, we expect that a regression will produce values near 0.5 for \( a \) and 1.5 for \( b \) (1 from the additive 1 plus .5 for the average value returned by RAND). In
fact, we obtain

```
1 XCOL 2 YCOL LINFIT LR
   Intercept: 1.51222496525  Slope: .500206997492
```

In this example, we are using uniformly distributed noise, which is easy to generate using RAND (section 20.5.2). In the next section, we give an example of a regression applied to normally distributed pseudorandom numbers.

Once you have computed the regression coefficients, you can use them to compute estimates of \( y \) for values of \( x \) that are not in the data set. This is achieved by PREDY (\texttt{PREDicted Y}), which takes an \( x \) value from the stack and returns a \( y \) value computed from the coefficients stored in \texttt{ΣPAR}. For example, using the data from the current example,

```
200 PREDY \( \Leftrightarrow \) 101.553624464
```

The HP 48 also provides PREDX, which computes a value of \( x \) given a value of \( y \):

```
60 PREDX \( \Leftrightarrow \) 116.927142819
```

Keep in mind, however, that \( x \) is the independent variable and may not even be continuously variable, so that PREDX's result may not be entirely meaningful. Note also that PREDX computes \( x = (y - b)/a \) using the coefficients computed from the regression of \( y \) on \( x \). This is different in principle from the result that can be obtained by interchanging the roles of \( x \) and \( y \) and recomputing the regression coefficient.

### 20.3.2.1 Pseudo-Linear Regressions

The HP 48 stretches the idea of least-squares analysis a little by providing additional regression models in which the data is converted into a \textit{pseudo-linear} form by logarithmic transformations. Since this also distorts the noise associated with the variables, the use of a least-squares approach loses some validity since the simple linear method assumes at least that the noise has a constant variance. However, the accuracy of the resulting fits is likely to be more than sufficient for simple modeling of many physical or other systems.

The linear model and three pseudo-linear models, and the commands that activate each model, are as follows:
The four model selection commands (plus the automatic model selector BESTFIT) are found in a sub-menu activated by the MODL key in the fourth page of the statistics menu. Each of the four commands, like their plot type counterparts (section 15.2.8), executes by storing itself as the fifth element in the PAR list. LR checks that element to determine which type of transformation to perform prior to calculating a linear regression.

The logarithmic, exponential, and power model regressions are calculated by applying logarithms to the data to transform the model to a linear one, performing a linear regression on the transformed data, then inverse-transforming the computed parameters to the original model form. For example, for the logarithmic model, the data is presumed to have the form $y = a \ln x + b$. If we make the substitution $x' = \ln x$, then the model becomes $y = ax' + b$, which is the normal linear form. Similarly, the exponential and power forms can be transformed to linear by substituting for $x, y,$ and the linear coefficient $a$. The actual substitutions for the four models are shown here:

<table>
<thead>
<tr>
<th>Model</th>
<th>$a'$</th>
<th>$x'$</th>
<th>$y'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$a$</td>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>Exponential</td>
<td>$\ln a$</td>
<td>$x$</td>
<td>$\ln y$</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>$a$</td>
<td>$\ln x$</td>
<td>$y$</td>
</tr>
<tr>
<td>Power</td>
<td>$\ln a$</td>
<td>$\ln x$</td>
<td>$\ln y$</td>
</tr>
</tbody>
</table>

To illustrate a pseudo-linear regression, the following sequence creates data corresponding to a noisy logarithmic curve with $a=5$ and $b=1$, where $b$ is actually a normally distributed random variable with mean 1 and standard deviation 1. The independent variable values $x_i$ are the integers 1 through 100.
The data is stored in the variable \texttt{LOGDAT}. Then, \texttt{LOGDAT} \texttt{STO LOGFIT LR} \Rightarrow \texttt{Intercept: 1.5848798893 Slope: 4.90171452.}

This data and the computed regression line are plotted in section 15.9.1.

\textit{Example}. In a high-school chemistry class experiment, the students measure the decay of $\text{Ba}^{137}$ using a geiger counter. At 100 second intervals, they record the following count rates, after subtracting the background rate:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Interval} & \textbf{Rate} \\
\hline
1 & 1611 \\
2 & 1007 \\
3 & 657 \\
4 & 510 \\
5 & 306 \\
6 & 198 \\
7 & 113 \\
8 & 90 \\
9 & 44 \\
10 & 29 \\
\hline
\end{tabular}
\end{center}

Find the half-life of the isotope from this data.

\textit{Solution}. Radioactive substances exhibit an exponential decay in their radioactivity

\[ R(t) = R(0)e^{-\frac{t}{\lambda}} \]

where \( \lambda \) is the half life. This formula matches the HP48 exponential fit model, where \( a = -\ln(2)/\lambda \) and \( b = R(0) \). Entering the data in the above table into \texttt{ΣDAT}, we find

\[ 3 \text{ FIX EXPFIT LR} \Rightarrow \texttt{Intercept: 2626.418 Slope: -0.442} \]
The half-life is derived from the slope:

\[ \frac{2 \ln \text{NEG} \text{SWAP}}{1} = 1.567 \]

This result is in units of 100 seconds, so that the calculated half-life is actually 156.7 seconds.

### 20.3.3 Best Fit

**BESTFIT** is a variation of LR which performs a regression for each of the four models, and chooses the best fit from among them. The coefficients from that best fit, and the name of the corresponding model selection command are stored in ZPAR, but no objects are returned to the stack.

"Best" in this context refers to the model for which linear regression returns the largest correlation coefficient. For the first example in the previous section, separate execution of LR for each of the four models yields the following correlations:

<table>
<thead>
<tr>
<th>Model</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.871</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>0.976</td>
</tr>
<tr>
<td>Exponential</td>
<td>0.636</td>
</tr>
<tr>
<td>Power</td>
<td>0.863</td>
</tr>
</tbody>
</table>

From these results it is evident that the logarithmic model provides the best fit, and indeed **BESTFIT** returns **LOGFIT** and the coefficients derived from that model to ZPAR.

A purist might question the value of a "best fit," since the use of curve fitting at all implies that you already know the nature of the curve you are fitting to the data. The regression process should be to determine the unknown parameters of the model, not the model itself. Despite these reservations, finding a best-fit curve can be quite useful, especially when you are attempting to find an analytic expression to approximate data that isn't derived from any simple theoretical model, like the price of a security or a sales forecast.

### 20.3.4 Scatter Plots with Error Bars

An ordinary scatter plot (section 15.9) plots each point as a single dark pixel. It is common in experimental measurement problems to represent the error in a given measurement \((x_i, y_i)\) by drawing a vertical error bar through its plotted point, from \(y_i + \sigma_i\) to \(y_i - \sigma_i\), where \(\sigma_i\) is the standard deviation of the \(i\)th measurement. In the next section,
where we discuss weighted measurements, such plots can help to verify the results of a least-squares fit. But even in unweighted cases, you may find it useful to plot with small error bars to make the plotted points stand out against a fitted curve.

The program DRAWEBAR listed below is a variation of DRAW that makes a scatter plot with error bars. Its arguments are a measurement vector $y$ (level 3), an independent variable vector or $n \times 1$ matrix $x$, and an object that specifies the half-height of the bars. The latter object can be a real number $\sigma$, or a real vector comprised of the different $\sigma_i$'s for each point.

For example, for the barium decay problem in the previous section, the measurement error should be equal to the square root of each count, since radioactive decay rates follow a Poisson distribution. To create a plot that reflects these errors, execute the following:

```
RCLy CLSPLIT OVER OBJ-
DROP 1 10 START V 10 ROLL NEXT 10 -ARRY
```

This creates a vector from the standard deviations. (CLSPPLIT is listed in section 20.4.1.)

Then

```
-1 11 XRNG -200 1800 YRNG DRAWEBAR
```

makes this plot:

These bars are relatively small, so you would expect a good fit of a logarithmic curve to the data.
20.3.5 Summary Statistics

The last page of the STAT menu contains commands that return one- and two-variable summary statistics from ΣDAT. Each of these commands returns a real number representing a sum:

<table>
<thead>
<tr>
<th>Command</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΣX</td>
<td>( \sum x_i )</td>
</tr>
<tr>
<td>ΣY</td>
<td>( \sum y_i )</td>
</tr>
<tr>
<td>ΣX^2</td>
<td>( \sum x_i^2 )</td>
</tr>
<tr>
<td>ΣY^2</td>
<td>( \sum y_i^2 )</td>
</tr>
<tr>
<td>ΣX*Y</td>
<td>( \sum x_i y_i )</td>
</tr>
<tr>
<td>NΣ</td>
<td>( n )</td>
</tr>
</tbody>
</table>

where the \( x_i \) are the elements in the independent column, \( y_i \) are the elements in the dependent column, and \( n \) is the number of rows in ΣDAT.

When ΣDAT contains \( m \geq 2 \) columns, you can obtain all of the summed squares and
cross products by premultiplying the data matrix by its own transpose. You can also obtain the summed columns as well by grafting a column of 1's to either side of the matrix:

\[
\Sigma\text{DAT} \ \text{CLADD1 \ DUP \ TRN \ SWAP \ *} 
\]

(CLADD1 is defined in section 20.4.1.) The resulting \(m+1 \times m+1\) square matrix \(\mathbf{c}\) has the following elements, where the \(x_{ij}\) are the elements of \(\Sigma\text{DAT}\):

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_{ij}) ((i,j \leq m))</td>
<td>(\sum_{k=1}^{n} x_{ki}x_{kj})</td>
</tr>
<tr>
<td>(c_{m+1,j}) ((j \leq m))</td>
<td>(\sum_{k=1}^{n} x_{kj})</td>
</tr>
<tr>
<td>(c_{j,m+1}) ((j \leq m))</td>
<td>(\sum_{k=1}^{n} x_{kj})</td>
</tr>
<tr>
<td>(c_{m+1,m+1})</td>
<td>(n)</td>
</tr>
</tbody>
</table>

The sums obtained this way are not as accurate as those obtained with \(\Sigma\mathbf{X}\), etc., since the built-in commands accumulate the sums and products using 15-digit internal precision.

### 20.4 General Least-Squares Fitting

The linear regressions described in the preceding sections are simple cases of the application of least-squares fitting. Given the HP 48's powerful array-handling capabilities, it is straightforward to program the calculator to handle a wider variety of curve fitting problems than is provided by the LR command.

The principles of the least-squares method summarized below are presented without proof. You are referred to any standard statistics or data analysis textbook for a more complete discussion. One such source, which has a strong emphasis on numerical analysis, is *Applied Numerical Methods*, by Carnahan, Luther, and Wilkes (John Wiley & Sons, 1969).

The basic purpose of any curve fitting is to determine the values of parameters that can't be measured directly, from measurable quantities that are functions of those parameters. Initially, we will consider the measurement of a variable \(\phi\) that is a linear function of \(m\) independent variables \(x_j\):
\[ \phi = \sum_{j=1}^{m} x_j a_j \]

where the coefficients \( a_j \) are the parameters that we are to determine. Assume that we have a set of \( n \) measurements \( y_i \), taken for various specific values \( x_{ij} \) of the independent variables. The differences \( y_i - \phi(x_{ij}) \) are assumed to be random variables, with standard deviations \( \sigma_i \). We then want to compute values of the \( a_j \) such that the sum of the weighted squared errors is minimized:

\[ E = \sum_{i=1}^{n} \frac{(y_i - \sum_{j=1}^{m} x_{ij} a_j)^2}{\sigma_i^2}. \]

The notation is simplified if we represent the \( y_i \) as an \( n \)-element column vector \( y \), the \( a_j \) as an \( m \)-element column vector \( a \), and the \( x_{ij} \) as an \( n \times m \) matrix \( X \). The weights \( 1/\sigma_i^2 \) can be combined into a \( n \times n \) diagonal matrix \( W \), with \( W_{ii} = 1/\sigma_i^2 \). Then the least-square error computation is to minimize

\[ E = (y - Xa)^T W (y - Xa) \]

To find the minimum, we differentiate with respect to each of the \( a_j \), and set the results equal to zero. The resulting set of simultaneous equations is

\[ 2X^T W (y - Xa) = 0. \]

Solving for \( a \),

\[ a = (X^T W X)^{-1} X^T W y \]

Notice that if the \( \sigma_i \) are all the same (such as is assumed in the LR linear regressions), \( W \) is a multiple of the identity matrix and so cancels out of this equation. The standard deviations of the parameters \( a \) are the diagonal elements of the covariance matrix

\[ V = (X^T W X)^{-1}. \]

If \( \sigma_i = \sigma = constant \), then

\[ V = (X^T X)^{-1} \sigma^2. \]

Moreover, if the \( \sigma_i \) are not known a priori, but are assumed to be uniform, then that standard deviation can be estimated from

\[ \sigma^2 = \frac{y^T y - y^T X a}{n - m} \]

which is the averaged squared difference between \( y_i \) and \( (Xa)_i \).
The program LSFIT listed next embodies the principles outlined here:

<table>
<thead>
<tr>
<th>LSFIT</th>
<th>Least-Squares Fit</th>
<th>B3B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{align*}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>level 3 &amp; level 2 &amp; level 1 &amp;</td>
<td>level 3 &amp; level 2 &amp; level 1 &amp;</td>
<td></td>
</tr>
<tr>
<td>y &amp; X &amp; W &amp; \sigma &amp; a &amp; V &amp; \sigma &amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\end{align*}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{\textlangle{}} \text{RO} \text{T} \text{DU} \text{P} \text{ SIZE} & \ 1 \ + \ 1 \ 2 \ \text{SUB} \ \text{RDM} \\
\text{RO} \text{T} \text{DU} \text{P} \text{ SIZE} & \ 1 \ + \ 1 \ 2 \ \text{SUB} \ \text{RDM} \\
\text{DU} \text{P} \ \text{SIZE} & \ \text{OBJ} - \rightarrow \ w \ y \ x \ n \ m \ w? \\
\text{\textlangle{}} \ \text{CASE} \ w & \ 0 \ \text{SAME} \\
\text{THEN} \ 1 \ \text{'w'} \ \text{STO} & \ 0 \ \text{'w'}? \ \text{STO} \ \text{END} \\
\text{w} & \ \text{TYPE} \\
\text{THEN} \ w & \ \text{SIZE} \ \text{SIZE} \\
\text{IF} & \ 1 \ \text{SAME} \\
\text{THEN} \ n & \ \text{IDN} \ 1 \ n \\
\text{FOR} \ i & \ i \ i \ 2 \ \text{-LIST} \ w \ i \ \text{GET} \\
\text{SQ} & \ \text{INV} \ \text{PUT} \\
\text{NEXT} \ \text{'w'} & \ \text{STO} \\
\text{END} \\
\text{END} \\
\text{w} & \ \text{SQ} \ \text{INV} \ \text{'w'} \ \text{STO} \\
\text{END} \\
\text{x} & \ \text{TRN} \ w & \ \text{*} \ \text{x} & \ \text{*} \ \text{INV} \\
\text{DU} \text{P} \ \text{x} & \ \text{TRN} & \ \text{*} & \ w & \ \text{*} & \ y & \ \text{*} \\
\text{DU} \text{P} & \ m & \ 1 \ \text{-LIST} \ \text{RDM} \ 3 \ \text{ROLLD} \\
y & \ \text{x} \ \text{RO} \text{T} & \ \text{-} \ \text{DUP} \ \text{TRN} \ \text{SWAP} & \ \text{*} \\
1 & \ \text{GET} & \ n & \ m & \ - & \ / \\
\text{IF} & \ w? & \ \text{NOT} \\
\text{THEN} & \ \text{SWAP} & \ \text{OVER} & \ \text{*} \ \text{SWAP} \\
\text{END} & \ \sqrt{ } \\
\text{\textrangle{}} \\
\end{align*}
\]

Convert y to a $n \times 1$ array.
Likewise for x.

\text{w? true} means weights are supplied.

Unweighted case.
If w is not a real number,
then if it's a vector of $\sigma$'s
remake into a diagonal matrix:

\[
\sigma_i \ 1/\sigma_i^2
\]

w is a number; replace with $1/\sigma_i^2$.

Covariance matrix V.
Parameter vector a.

Convert a to a vector.

Estimate $\sigma^2$
If unweighted,
then multiply V by $\sigma^2$
Return $\sigma$.

LSFIT takes the following arguments:

3: y Vector or $n \times 1$ matrix
2: X Vector or matrix
1: W Real number, vector, or square matrix

For cases of uniformly weighted measurements, W is entered as a real number $\sigma$ that is the standard deviation of each measurement. For non-uniform weights, W may be
entered as a vector of $\sigma_i$’s or as a square diagonal matrix where $W_{ii} = 1/\sigma_i^2$. Entering 0 for $W$ indicates unweighted measurements.

LSFIT returns the following:

3: $a$ Vector of computed parameters
2: $V$ Covariance matrix
1: $\sigma$ Standard deviation

The standard deviations of the parameters $a_i$ are given by the square roots of the diagonal elements of the covariance matrix; $\sigma$ is the average deviation of each $y_i$ from the fitted curve.

**Example.** Find the parameters $a$ and $b$ that provide the best fit of the following data to the function $ax^2 + b$:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12.25</td>
</tr>
<tr>
<td>4</td>
<td>7.35</td>
</tr>
<tr>
<td>3</td>
<td>2.59</td>
</tr>
<tr>
<td>2</td>
<td>-.28</td>
</tr>
<tr>
<td>1</td>
<td>-1.75</td>
</tr>
<tr>
<td>0</td>
<td>-2.88</td>
</tr>
<tr>
<td>-1</td>
<td>-1.64</td>
</tr>
<tr>
<td>-2</td>
<td>.14</td>
</tr>
<tr>
<td>-3</td>
<td>2.42</td>
</tr>
<tr>
<td>-4</td>
<td>7.24</td>
</tr>
<tr>
<td>-5</td>
<td>12.55</td>
</tr>
</tbody>
</table>

**Solution.** In this problem, the independent variables corresponding to the unknown coefficients are $x^2$ and the constant 1. First, enter the vector $y$:

$$[12.25 \ 7.35 \ 2.59 \ -0.28 \ -1.75 \ -2.88 \ -1.64 \ 0.14 \ 2.42 \ 7.24 \ 12.55]$$

Next, enter the matrix $X$:

$$[[25 \ 1] \ [16 \ 1] \ [9 \ 1] \ [4 \ 1] \ [1 \ 1] \ [0 \ 1]$$

$$[1 \ 1] \ [4 \ 1] \ [9 \ 1] \ [15 \ 1] \ [25 \ 1]]$$
Finally, enter the weight matrix, in this case a vector containing the $\sigma_i$:

$$[ .1 \ 2 \ .2 \ .3 \ .3 \ .4 \ .4 \ .4 \ .3 \ .3 \ .2 ]$$

Then

$$2 \text{ FIX LSFIT }$$

$$\begin{bmatrix} 0.60 & -2.54 \\ 5.12E-5 & -8.85E-4 \\ -8.85E-4 & 0.02 \end{bmatrix} 0.30$$

The first result is the coefficient vector, corresponding to $a=0.6$, and $b=-2.54$. The second result is the correlation matrix; looking at the square roots of the diagonal elements we can estimate $\sigma_a = 0.007$ and $\sigma_b = 0.14$. The last result is the average error at each point, which matches reasonably well with the standard deviations of the individual data points.

### 20.4.1 Utilities

In the previous example, we entered the $y$, $X$, and $\sigma$ arrays individually. For manual entry, it may be more convenient to enter the data point-by-point using $\Sigma +$. That is, for each point $i$, the entry for $\Sigma +$ might have the form

$$x_{i1} \ \cdots \ x_{im} \ y_i \ \sigma_i$$

Once the data is accumulated into $\Sigma$DAT, it needs to be decomposed into the three separate arrays for LSFIT. The following two programs are utilities that are useful for that purpose. **C1SPLIT** removes the first column of an $n \times m$ array, returning the column as a vector and the remainder as an $n \times m - 1$ matrix. **CLSPLIT** works similarly, removing the last column.

<table>
<thead>
<tr>
<th>C1SPLIT</th>
<th>Column 1 Split</th>
<th>ADD9</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 1</td>
<td>level 2</td>
<td>level 1</td>
</tr>
<tr>
<td>$[[n \times m]]$</td>
<td>$\sigma$</td>
<td>$[n]$</td>
</tr>
</tbody>
</table>

| $\leftarrow \ TRN \ OBJ- \ OBJ- \ DROP \rightarrow \ m \ n$ | Take data matrix apart. |
| $\leftarrow m \ 1 \ - \ n \ 2 \ - \ LIST \ - \ ARRY \ TRN$ | Recombine last $m-1$ rows. |
| $n \ 1 \ + \ ROLLD \ n \ - \ ARRY \ SWAP$ | Make first row into a vector. |

| $\leftarrow \ TRN \ OBJ- \ OBJ- \ DROP \rightarrow \ m \ n$ |
| $\leftarrow m \ 1 \ - \ n \ 2 \ - \ LIST \ - \ ARRY \ TRN$ |
| $n \ 1 \ + \ ROLLD \ n \ - \ ARRY \ SWAP$ |
For cases where one of the unknown coefficients is simply an additive constant, using one of the following two programs will save you from having to enter a 1 in each row of the data matrix. C1ADD1 prepends a column of 1's to an array; CLADD1 appends a column of 1's.

### C1ADD1

**Column 1 Added with 1's**

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[n \times m]]$</td>
<td>$[[n \times (m+1)]]$</td>
</tr>
</tbody>
</table>

$$\left< \begin{array}{c} \text{TRN} \ \text{DUP} \ \text{SIZE} \ \text{OBJ-} \ \text{DROP} \ - \ m \ \ n \\ \text{n} \ + \ 1 \ - \ \text{LIST} \ \text{CON} \ \text{OBJ-} \ \text{DROP} \\ \text{n} \ + \ \text{ROLL} \ \text{OBJ-} \ \text{DROP} \\ \text{m} \ + \ n \ 2 \ - \ \text{LIST} \ - \ \text{ARRY} \ \text{TRN} \end{array} \right>$$

Save dimensions.

Enter n 1's.

Retrieve original matrix.

Combine into new matrix.

### CLADD1

**Column Last Added with 1's**

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[[n \times m]]$</td>
<td>$[[n \times (m+1)]]$</td>
</tr>
</tbody>
</table>

$$\left< \begin{array}{c} \text{TRN} \ \text{OBJ-} \ \text{OBJ-} \ \text{DROP} \ - \ m \ \ n \\ \text{n} \ - \ \text{LIST} \ 1 \ \text{CON} \ \text{OBJ-} \ \text{DROP} \\ \text{m} \ + \ n \ 2 \ - \ \text{LIST} \ - \ \text{ARRY} \ \text{TRN} \end{array} \right>$$

Save dimensions.

Enter n 1's.

Combine with original matrix.
With these utilities available, the data for the example in the previous section can be entered into ΣDAT:

```
CLΣ [ 25 12.25 0.1 ] Σ+
16 7.35 0.2 Σ+
9 2.59 0.2 Σ+
4 -.28 0.3 Σ+
1 -1.75 0.3 Σ+
0 -2.88 0.4 Σ+
1 -1.64 0.4 Σ+
4 .14 0.4 Σ+
9 2.42 0.3 Σ+
16 7.24 0.3 Σ+
25 12.55 0.2 Σ+
```

Then

```
ΣDAT CLSPLIT CLSPLIT CLADD1 ROT
```

sets up the stack arguments for LSFIT.

### 20.4.2 Polynomial Fits

A common curve-fitting problem is that of fitting data to a polynomial, where there is a single independent variable $x$, and the fit parameters are the coefficients of successive powers of $x$. LSFIT is easily adapted to this problem by supplying a matrix $X$ for which the elements are $x_{ij} = x^{m-j}$. The width $m$ of the matrix is one greater than the highest power of $x$ that appears in the polynomial.

POLYFIT is an extension of LSFIT specifically for polynomials. It takes the same arguments as LSFIT, plus a real number that specifies the order of the polynomial. The the matrix $X$ should be a vector or a one-column matrix, where each element is one value of the independent variable $x$. POLYFIT creates a new $X$ by computing the necessary powers of each $x_i$. POLYFIT concludes by calling LSFIT with the modified arguments. Note that the (level 3) result $a$ contains the polynomial coefficients in the same descending order that is expected by the polynomial programs in section 17.4.

- **Example.** Fit a fourth-degree polynomial to the data in the following table, where the standard deviation of each measurement is 0.85:

```
\[ x \quad y \\
5 \quad 34.8 \\
10 \quad 134.7 \\
14 \quad 159.3 \\
18 \quad 156.9 \\
24 \quad 132.2 \\
30 \quad 117.4 \\
35 \quad 132.5 \\
41 \quad 186.6 \\
50 \quad 342.2 \\
\]

(This problem is taken from the *HP 82484A Curve Fitting Pac Owner's Manual* for the HP71.)

**Solution.** Enter \( y \):

\[
\begin{bmatrix}
34.8 & 134.7 & 159.3 & 156.9 & 132.2 & 117.4 & 132.5 & 186.6 & 342.2
\end{bmatrix}
\]

Enter \( X \):

\[
\begin{bmatrix}
5 & 10 & 14 & 18 & 24 & 30 & 35 & 41 & 50
\end{bmatrix}
\]

Enter \( \sigma \) and the order, and compute the fit:

\[ .85 \quad 4 \quad \text{POLYFIT}. \]

The level 3 result vector contains the following entries, to four places:

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_4 )</td>
<td>-0.0004781</td>
<td>0.000011</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>0.07049</td>
<td>0.0013</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>-3.270</td>
<td>0.047</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>57.48</td>
<td>0.66</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>-179.2</td>
<td>2.9</td>
</tr>
</tbody>
</table>

The standard deviations are the square roots of the diagonal elements of the level 2 result matrix. The calculation returns (level 1) \( \sigma = 0.811 \), which is comparable to the presumed standard deviation of 0.85.
20.4.3 Non-Linear Least Squares Fits

The least-squares fitting process can be extended to non-linear cases, i.e. cases where the fit parameters \( a_j \) are not simple linear coefficients of the independent variables. That is, the model to be fitted to measured data has the general form \( \phi = \phi(x, a) \). We can relate this general case to the linear problem discussed in section 20.4 by expanding \( \phi \) about the point \( a_0 \):

\[
\phi(x,a) \approx \phi(x,a_0) + \Delta \phi \cdot \Delta a
\]

where

\[
\Delta \phi_j = \frac{\partial \phi}{\partial a_j}(x,a_0)
\]

and \( \Delta a = a - a_0 \). This is linear in the parameters \( \Delta a \), and so can be translated to the linear least-squares problem \( \phi = Xa \) by replacing \( \phi \) with \( \phi(x,a) - \phi(x,a_0) \), \( X \) with \( \Delta \phi \), and \( a \) with \( \Delta a \).

Therefore, if we take \( a_0 \) as a first guess for \( a \), and compute \( \Delta a \) by the least squares method, then \( a + \Delta a \) should provide a better estimate of \( a \). This process can be iterated until \( \Delta a \) is arbitrarily small. The trick, of course, is to obtain initial values \( a_0 \) that are
sufficiently accurate for the linear approximation to be valid. With initial values that are too far from the correct values, the iteration may never converge.

The program NLFIT listed below demonstrates an iterative non-linear least-squares fit, for problems with a single independent variable \( x \). It requires six arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6: ( y )</td>
<td>Vector of original measurements ( y_i )</td>
</tr>
<tr>
<td>5: ( x )</td>
<td>Vector of values ( x_i ) of the independent variable.</td>
</tr>
<tr>
<td>4: ( W )</td>
<td>Weights, entered as for LSFIT.</td>
</tr>
<tr>
<td>3: ( \phi(x,a) )</td>
<td>Algebraic object representing ( \phi ).</td>
</tr>
<tr>
<td>2: ( { a_i } )</td>
<td>List of parameter names.</td>
</tr>
<tr>
<td>1: ( { a_{i0} } )</td>
<td>Vector of initial values of parameters.</td>
</tr>
</tbody>
</table>

To simplify the program itself and argument entry, NLFIT uses the specific name \( X \) to represent the independent variable \( x \) (you can easily modify the program to use any other global name). This means that the expression entered as the level three argument should contain \( X \), as well as the names of the fit parameters. NLFIT uses the actual variables during program execution, so any previous stored values are overwritten.

After computing one iteration of the non-linear least squares fit, NLFIT halts and displays the corrections \( \Delta a \) (level 2) and the average error per point \( \sigma \) (level 1). You can use that information to decide whether to repeat the iteration. Use \( \text{MORE} \) in the displayed temporary menu to continue with another iteration, or \( \text{QUIT} \) to terminate the program and return the final results, which are the parameter vector \( a \) (level 3), the covariance matrix \( V \) (level 2), and the final \( \sigma \). The two objects returned at each intermediate halt are not required by the program, so you can drop them, use them in other calculations, or leave them for comparison with the results of the next iteration. If you change menus, you can resume execution by pressing 0 [CONT] for \( \text{MORE} \) , or 1 [CONT] for \( \text{QUIT} \).

The use of NLFIT is best explained by an example.

**Example.** The data in the table below (following the program listing) was obtained by measurement of a voltage at uniform time intervals. The voltage is expected to behave sinusoidally with time, i.e.

\[
V(t) = V_0 + A \sin(\omega t + \theta)
\]

Find the unknown parameters \( V_0, A, \omega \) and \( \theta \), which are estimated to be \( V_0 = 9 \) volts, \( A = 11 \) volts, \( \omega = .6 \), and \( \theta = .9 \). The measurement error is \( \pm 1.75 \) volts.
Create local variables.

Store initial values of \( \mathbf{a} \).

Prevent evaluation of \( \mathbf{X} \).

Create list of derivatives:

Store \( \mathbf{x}_i \) in \( \mathbf{X} \).

For each data point:

Store \( \mathbf{x}_i \) in \( \mathbf{X} \).

For each parameter:

Make data array.

Compute, save fit parameters.

Corrected \( \mathbf{a} \).

Halt and show results.

If done,

Store final values of \( \mathbf{a} \),

and return fit results.

Otherwise, repeat.

Restore original menu.
### Solution

1. Enter the measurement vector:

   \[
   \begin{bmatrix}
   -0.32 & 3.93 & 5.76 & 7.02 & 13.81 & 19.33 & 22.86 & 22.91 \\
   22.02 & 21.86 & 13.35 & 12.22 & 8.07
   \end{bmatrix}
   \]

2. Enter the vector of independent variable values. Either type it in, or execute

   \[
   0 \ 6 \ FOR \ i \ i \ .3 \ STEP
   \]

3. Enter 1.75 for the measurement error.

The objects on the stack at this point are also those needed to plot the data using DRAWBAR (section 20.3.4):

\[
-1 \ 7 \ XRNG \ -4 \ 24 \ YRNG \ 3 \ DUPN \ ERASE \ DRAWBAR
\]
4. Enter the fit expression:

\[ V_0 + A \sin(\omega \times X + \theta) \]

Notice that the independent variable must be \( X \). \( \omega \) is \( \alpha \rightarrow W \), and \( \theta \) is \( \alpha \rightarrow F \).

So that you can plot this expression later, execute DUP STEQ.

5. Specify the unknown parameter names:

\( \{ V_0 \ A \ \omega \ \theta \} \)

6. Enter the initial guesses for the parameters:

\[ [9 \ 11 \ 1.6 \ 9] \]

7. Execute RAD 2 FIX NLFIT. This produces the following display:

```
RAD HALT
\{ HOME STAT REGR \}
4:
3:
2: [ 3.48 -1.16 -0.10...
1: \( \sigma \): 1.74
MORE QUIT
```

The increments \( \Delta a \) returned to level 2 are substantial fractions of the initial guesses, which suggests continuing with the iteration. Press MORE:

```
RAD HALT
\{ HOME STAT REGR \}
4: [ 3.48 -1.16 -0.10...
3: \( \sigma \): 1.74
2: [ 0.02 0.34 -0.01...
1: \( \sigma \): 1.75
MORE QUIT
```
This iteration has not changed \( \sigma \) very much, so further iteration is unlikely to produce a better result. Now press \texttt{QUIT}:

![Statistics Menu](image)

The final values \( a \) are also stored in the appropriate global variables. This means that you can plot the expression stored in \texttt{EQ} to show it as evaluated with the fit parameters, superimposed on the data:

![Plot](image)

### 20.5 Probability Commands

The probability menu (\texttt{[MTH] PROB}) contains nine commands that relate to common probability calculations. The commands logically could have been included as part of the statistics menu, but that menu is already five pages long.

#### 20.5.1 Combinations and Permutations

The commands \texttt{COMB} and \texttt{PERM} compute the number of \textit{combinations} and \textit{permutations}, respectively, of \( n \) different objects taken \( m \) at a time. Each different ordering of the \( m \) selected objects is a different permutation. Since there are \( n \) choices for the first object, and \( n-1 \) choices for the second, and so on, we obtain
\[ P_{n,m} = n(n-1)(n-2) \cdots (n-m+1) = \frac{n!}{(n-m)!} \]

PERM takes its arguments in the order \( n,m \):

\[ 10 \ 5 \ \text{PERM} \ = \ 30240, \]

or

\[ \text{PERM}(10,5) \ \text{EVAL} \ = \ 30240. \]

For combinations, the order of the selected objects doesn't matter; since there are \( m! \) ways to arrange \( m \) objects, the number of combinations is reduced by that factor from the number of permutations:

\[ C_{n,m} = \frac{P_{n,m}}{m!} = \frac{n!}{m!(n-m)!}. \]

COMB takes its arguments in the same order as PERM:

\[ 10 \ 5 \ \text{COMB} \ = \ 252 \]

The \( C_{n,m} \) are the same as the binomial coefficients \( \binom{n}{m} \) in the expansion

\[ (x + y)^n = \sum_{m=1}^{n} \binom{n}{m} x^{n-m} y^m. \]

COMB is therefore used to compute the binomial coefficients in the program PPWR, in section 17.4.1.

The algorithms used in computing COMB and PERM do not use the factorial function directly, to minimize numerical overflow problems. Factorial overflows for integer arguments greater than 253, but COMB and PERM can deal with arguments up to \( 10^{12} \). Arguments within that range can still cause the overflow exception (section 9.6.4) when the results are greater than \( 10^{500} \). In general, both arguments for either command must be zero or positive integers less than \( 10^{12} \). Non-integer or too-large arguments return the Bad Argument Value error.
20.5.1.1 Factorial and Gamma Function

The HP48 factorial function ! computes the ordinary factorial of integer arguments \( n \geq 0 \): \( n! = n(n-1)(n-2) \cdots 2 \cdot 1 \). However, ! is actually a version of the gamma function:

\[
\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} \, dt.
\]

That is, since \( \Gamma(x+1) = x! \), HP 48 \( x! \) computes \( \Gamma(x+1) \). If you want an ordinary gamma function, create a user-defined function as follows:

\[
'\text{GAMMA}(x) = (x-1)!' \quad \text{DEFINE}
\]

The useful range of ! is \(-254.1082426465 < x < 253.1190554375\). More negative \( x \) causes the underflow exception; more positive \( x \) causes the overflow exception. Negative integer values of \( x \) also cause overflow.

The factorial is represented as a postfix function for consistency with common written notation. You may also enter it as a prefix function \( \text{FACT} \), which is automatically converted to the postfix ! when it is evaluated:

\[
'\text{FACT}(X)' \quad \text{EVAL} \quad 'X!'
\]

assuming that \( X \) has no current value.

20.5.2 Random Numbers

Several examples in this chapter have used the random number generator command \( \text{RAND} \) to create artificial "noise" in data. \( \text{RAND} \) is designed to produce random numbers uniformly distributed between 0 and 1, meaning that any number in that range is equally likely to be returned at each execution. Since there are \( 10^{12} - 1 \) HP 48-representable numbers in this range, the probability density is

\[
\rho_n = 1/(10^{12} - 1) \quad 0 < n < 1
\]

The expectation value of the average of a sample of numbers generated by \( \text{RAND} \) is obviously 0.5. We can also estimate the standard deviation by treating the distribution function as continuous:

\[
\sigma = \left( \int_0^1 (x-.5)^2 \, dx \right)^{1/2} = \frac{1}{2\sqrt{3}}.
\]
RAND uses the *linear congruential method* to create uniform random numbers (cf. Donald Knuth, *The Art of Computer Programming* Vol. II, Addison-Wesley, Massachusetts 1981). In this method, a sequence of random numbers is generated from

\[ x_i = (ax_{i-1} + c) \mod m \]

Such sequences inevitably repeat, so that they are not truly random. The period of the cycles can be maximized by good choices of the constants \( a, c, \) and \( m \). RAND uses values that give a period of \( 5 \times 10^{13} \), using 15-digit internal precision.

The last value returned by RAND is stored in 15-digit form in system memory, so that it can be used for the next execution. You can change this "seed" value by executing RDZ (RanDomize), which computes and stores a new 15-digit seed from its argument, which can be any real number. RDZ is useful when you want to repeat a particular series of random numbers--you can restart the same series whenever you want by executing RDZ with the same argument. The examples in this book use RDZ prior to repeated execution of RAND, so that you can try the examples and get the same results. You can also use RDZ to prevent accidental repeat of a series, by using an argument of 0. In that case, RDZ creates a random seed based on a reading of the HP 48 clock.

As a rough check of the uniform distribution of the numbers generated by RAND, we can compute the mean and standard deviation of a sample of 1000 numbers:

```
12345 RDZ 1 1000 START RAND NEXT \{1000 1\} ->ARRY STO\Sigma,
```

then

```
MEAN \[\.507100107091\]
STD \[.289199195881\]
```

These compare well with the ideal values of 0.5 and \( .288676134595 \) (\( 1/2\sqrt{3} \)). The RAND algorithm also passes the *spectral test* (described by Knuth), a much more stringent test of a random number generator.

It is possible to produce random numbers with non-uniform distributions by using functions of uniform random numbers. The programs in section 12.11.1 illustrate the generation of numbers that follow the normal and the Poisson distributions.
20.6 Upper-Tail Probability Distributions

A random variable is characterized by its *distribution function*, which is the (cumulative) probability $P(x)$ that a particular measurement of the variable will have a value less than $x$. $P(x)$ must be a monotonically increasing function of $x$ (a step function for a discrete variable), with $P(\infty) = 1$. The complement to the distribution function is the *upper-tail distribution function* $U(x)$, which is the probability that the variable is measured with a value greater than $x$, i.e.

$$U(x) = 1 - P(x)$$

For continuous, differentiable distribution functions, the first derivative is called the *probability density* $\rho(x) = dP(x)/dx$; the probability that a measurement returns a value between $x$ and $x + dx$ is $\rho(x)dx$. The normalization condition for $\rho(x)$ is then

$$\int_{-\infty}^{+\infty} \rho(x)dx = 1.$$ 

Most probability distributions can be characterized most easily by the probability density, with the distribution functions calculated as integrals of the densities:

$$P(x) = \int_{-\infty}^{x} \rho(x)dx \quad \text{and} \quad U(x) = \int_{x}^{\infty} \rho(x)dx.$$ 

For example, the *normal*, or *Gaussian* distribution is defined by the density

$$\rho(x) = \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}},$$

where $\bar{x}$ is the average of $x$, and $\sigma$ is the standard deviation. Common probability densities generally have a long "tail" that asymptotically approaches zero as $x \to \infty$—this is the origin of the "upper tail" term that describes the HP 48 distribution functions.

Such densities are easily calculated on the HP 48 from their definitions; in section 20.6.5 we list user-defined function programs for the densities associated with the four HP 48 upper-tail probability commands. The distributions present a harder calculation problem since they involve numerical integrals with $\pm \infty$ as one limit, as shown here for the upper-tail probability of the normal distribution:

$$U(x) = \frac{1}{(2\pi)^{1/2}\sigma} \int_{x}^{\infty} e^{-\frac{(t-\bar{x})^2}{2\sigma^2}} dt$$

-673-
20.6 Statistics

Because of the difficulty of computing such integrals, the HP 48 provides four commands that use special algorithms for calculating upper-tail probabilities for common distributions. The upper-tail probabilities are chosen because they are used more frequently by statisticians than the distribution functions, but of course you can compute \( P(x) \) from \( 1 - U(x) \). The four commands are UTPN (Upper-Tail Probability--Normal), which embodies the formula above, plus UTPF, UTPC, and UTPT, for the \( F \)-, \( \chi^2 \)-, and \( t \)-distributions.

That these commands were not encoded as HP 48 functions was simply a matter of conserving ROM space in the HP 48. You can easily make user-defined functions for these commands, e.g.

\[
<< \rightarrow n1 \ n2 \ x \ << \ n1 \ n2 \ x \ UTPF \ >> \ >> 'FUTPF' \ \text{STO}
\]

### 20.6.1 UTPN

UTPN takes three real number arguments: the distribution average \( \bar{x} \), the variance \( \sigma^2 \), and the number \( x \). It returns the probability that a measurement of \( x \) will return a value greater than the specified \( x \), given a normal distribution described by \( \bar{x} \) and \( \sigma \).

- **Example.** On a standardized test with a mean score of 500 and a standard deviation of 100, what fraction of students is expected to score 750 or better, assuming a normal distribution of test scores?

- **Solution.** The variance is the square of the standard deviation, so

\[
500 \ 100 \ \text{SQ} \ 750 \ \text{UTPN} \ \Rightarrow \ .0062
\]

### 20.6.2 UTPT

The sample mean

\[
\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

(section 20.2) is itself a random variable. Its distribution is usually expressed in terms of the normalized parameter \( t = \frac{\bar{x} n^{1/2}}{s} \), where \( s \) is the square root of the sample variance. \( t \) is described by the probability density

\[
\rho(t) = \frac{\Gamma\left(\frac{f+1}{2}\right)}{\Gamma\left(\frac{f}{2}\right) \sqrt{f \pi}} \left(1 + \frac{t^2}{f}\right)^{-\frac{f+1}{2}},
\]
where \( f = n - 1 \) is the number of degrees of freedom.

UTPT takes \( f \) and \( t \) as its arguments, and returns the probability that a measurement of \( t \) will yield a value greater than the specified \( t \).

### 20.6.3 UTPC

The sample variance (section 20.2)

\[
s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

is itself a random variable. If the sample is drawn from a normal distribution with variance \( \sigma^2 \), then the random variable

\[
\chi^2 = \frac{n-1}{\sigma^2} s^2
\]

follows the so-called \( \chi^2 \)-distribution with \( f = n - 1 \) degrees of freedom. This distribution has the probability density

\[
\rho(\chi^2) = \frac{(\chi^2)^{\lambda-1} e^{-\chi^2/2}}{2^{\lambda} \Gamma(\lambda)}, \quad \chi^2 \geq 0
\]

where \( \lambda = f/2 \). The density is zero for \( \chi^2 < 0 \).

UTPC takes \( f \) and \( \chi^2 \) as its arguments, and returns the probability that a measurement of \( \chi^2 \) will yield a value greater than the specified \( \chi^2 \).

### 20.6.4 UTPF

The \( F \)-test compares the sample variances of populations that have equal means. The random variable in this case is the variance ratio

\[
F = \frac{s_1^2}{s_2^2}
\]

where \( s_1 \) and \( s_2 \) are the two sample variances, computed for \( n_1 \) and \( n_2 \) measurements, respectively.

The probability density for \( F \) is given by
\[ p(F) = \left( \frac{f_1}{f_2} \right) \frac{\Gamma\left(\frac{f_1 + f_2}{2}\right)}{\Gamma(f_1/2)\Gamma(f_2/2)} F^{\frac{f_1}{2} - 1} (1 + f_1 F/f_2)^{-\frac{f_1 + f_2}{2}}, \quad F \geq 0. \]

\[ = 0, \quad F < 0 \]

where \( f_1 = n_1 - 1 \) and \( f_2 = n_2 - 1 \) are the numbers of degrees of freedom of the two distributions.

UTPF takes \( f_1, f_2, \) and \( F \) as its arguments, and returns the probability that a measurement of \( F \) will yield a value greater than the specified value.
20.6.5 Probability Density Functions
The following user-defined functions compute probability densities for the normal, $\chi^2$, $t$, and $F$ distributions.

<table>
<thead>
<tr>
<th>NDIST</th>
<th>Normal Probability Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 3</td>
<td>level 2</td>
</tr>
<tr>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
</tr>
</tbody>
</table>

$$\sim \rightarrow m \sigma x '\text{EXP}(-((x-m)/\sigma)^2/2)/(\sqrt{2\pi}{\sigma})'$$

<table>
<thead>
<tr>
<th>CDIST</th>
<th>$\chi^2$ Probability Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 2</td>
<td>level 1</td>
</tr>
<tr>
<td>$f$</td>
<td>$x^2$</td>
</tr>
</tbody>
</table>

$$\sim \rightarrow f x '\text{IFTE}(x=0,x'(f/2-1)*\text{EXP}(-x/2)/(2^*(f/2)*(f/2-1)!),0)'$$

<table>
<thead>
<tr>
<th>TDIST</th>
<th>$t$ Probability Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 2</td>
<td>level 1</td>
</tr>
<tr>
<td>$f$</td>
<td>$t$</td>
</tr>
</tbody>
</table>

$$\sim \rightarrow f t '((f-1)/2)!/(f/2-1)!*\sqrt{f*(f+1)}*(1+t^2/f)^{-(f+1)/2}'$$

<table>
<thead>
<tr>
<th>FDIST</th>
<th>$F$-Distribution Probability Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 3</td>
<td>level 2</td>
</tr>
<tr>
<td>$f_1$</td>
<td>$f_2$</td>
</tr>
</tbody>
</table>

$$\sim \rightarrow f_1 f_2 F '\text{IFTE}(F\geq0,(f_1/f_2)^*(f_1/2)!/((f_1+f_2)/2-1)!/((f_1/2-1)! *((f_2/2-1)!)*F*(f_1/2-1)*(1+f_1/f_2*F)^{-(f_1+f_2)/2},0)'$$
21. Unit Management

A good deal of the effort involved in solving practical engineering and scientific problems comes from dealing with physical units. The results of most measurements are not simple dimensionless numbers, but combine a numerical magnitude with various units of length, time, mass, etc. Calculations with these values require keeping track of these units, including converting between values expressed with different units. Calculators have generally been unable to provide much assistance here. Many calculators contain tables of multiplicative conversion factors, which allow you to convert a number representing a measurement in one unit to another unit, such as multiplying a number of inches by 2.54 to obtain the equivalent number of centimeters. But the real accounting work here—collecting and canceling units and checking for dimensional consistency—is still left to pencil and paper (reminiscent of the pre-calculator era, where we used a slide-rule to perform operations on the mantissas of numbers, but had to keep track of the exponents by hand). The HP 48 removes this burden from you by providing a unit management system, incorporating physical units into a wide range of its calculation facilities.

In the HP 48, a magnitude and its associated units are combined into a single object called a unit object (section 3.4.9). These objects are entered and displayed in the format

\[ \text{magnitude}_\text{dimensions}, \]

where \textit{magnitude} is an ordinary real number, and \textit{dimensions} is an expression combining products, powers, and (at most) one quotient, of unit names. The magnitude is displayed in the same manner as a number within an algebraic object, following the current number display format (STD, FIX, etc.), except that digit separators are not used, and integers less than 1000 are displayed without any decimal part. The dimension part is displayed in a manner similar to an algebraic object. Since only products and simple powers are allowed, plus one quotient, parentheses are used only to enclose a denominator that contains a product.

When you enter a unit object in the command line, you can use same rules that are used in the object display. Note that you must enclose a compound denominator in parentheses, because of the usual algebraic operator precedence. That is,

\[ 1\text{_m/s}*s \quad \text{ENTER} \implies 1\text{_m}. \]

The entry is equivalent to \( 1_\text{(m/s)}*s \), so that the two \textit{s} units cancel each other.
Parentheses are also important when you enter a unit object within an algebraic object, to separate the unit object from the rest of the expression:

\[ 'Y+1\_m/s\times X' \quad \Rightarrow \quad 'Y+1\_m\times X/s' \]

\[ 'Y+(1\_m/s)\times X' \quad \Rightarrow \quad 'Y+(1\_m/s)\times X'. \]

Whether a unit object is entered in the command line, or created through operations on other unit objects, the dimension part is always simplified as follows:

- Any real numbers are multiplied together, then multiplied times the number part of the unit object. Thus 1_5*m becomes 5_m.

- There are only positive powers (any unit with a negative power is shown in the denominator).

- There is at most one / symbol; if the denominator is a product of units, it is enclosed in a single pair of parentheses.

- If a unit operation returns a result with a null dimensions, the result is converted to a real number. For example,

\[ 1\_m \quad 1\_m / \quad \Rightarrow \quad 1. \]

- Unit objects entered within expressions may be entered with symbolic magnitudes; such entries are automatically converted to the product of the symbolic magnitude times a normalized unit object. Thus 'A_m' becomes 'A*1_m'

### 21.1 Types of Units

The HP 48 unit system is based on the System Internationale, commonly abbreviated as SI. This system specifies base units, representing seven independent physical dimensions:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Base Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>meter</td>
</tr>
<tr>
<td>Time</td>
<td>second</td>
</tr>
<tr>
<td>Mass</td>
<td>kilogram</td>
</tr>
<tr>
<td>Temperature</td>
<td>kelvin</td>
</tr>
<tr>
<td>Electrical current</td>
<td>Ampere</td>
</tr>
<tr>
<td>Luminous intensity</td>
<td>candela</td>
</tr>
<tr>
<td>Quantity of substance</td>
<td>mole</td>
</tr>
</tbody>
</table>
These units are considered as fundamental; all other units are expressed in terms of the base units.

The HP 48 system actually uses eight base units. The extra dimension is unspecified, i.e. it is not related to any physical dimension, and so is reserved for use in user-defined units (see section 21.1.3 below). One built-in unit named ? has this dimension.

21.1.1 Prefixes
A natural extension to the built-in unit table is the *prefixing* of unit names with any of the following symbols:

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Word Prefix</th>
<th>SI symbol</th>
<th>HP 48 symbol(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+18</td>
<td>exa</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>+15</td>
<td>peta</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>+12</td>
<td>tera</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>+9</td>
<td>giga</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>+6</td>
<td>mega</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>+3</td>
<td>kilo</td>
<td>k</td>
<td>k or K</td>
</tr>
<tr>
<td>+2</td>
<td>hecto</td>
<td>h</td>
<td>h or H</td>
</tr>
<tr>
<td>+1</td>
<td>deka</td>
<td>da</td>
<td>D</td>
</tr>
<tr>
<td>-1</td>
<td>deci</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>-2</td>
<td>centi</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>-3</td>
<td>milli</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>-6</td>
<td>micro</td>
<td>μ</td>
<td>μ</td>
</tr>
<tr>
<td>-9</td>
<td>nano</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>-12</td>
<td>pico</td>
<td>p</td>
<td>p</td>
</tr>
<tr>
<td>-15</td>
<td>femto</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>-18</td>
<td>atto</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

These symbols are used immediately preceding the unit names they modify, without any intervening symbols or spaces. Thus 1_cm represents 1 *centimeter*, and 1_ft/ns represents 1 *foot/nanosecond*. Built-in unit names are given precedence over prefixed unit names when there is an ambiguity. For example, 1_min is 1 *minute*, not 1 *milli-inch*. This resolution requires single-character prefix symbols for simplicity, so the SI prefix *da* is replaced in the HP 48 by *D*. You may also use *K* for *k* and *H* for *h* when entering unit objects, but the objects will always be displayed with the lower-case prefixes.
21.1.2 Built-in Units

The HP 48 contains 122 units, which means that it has permanently stored their dimensions and conversion factors relative to the SI base units. For example, the unit mph is recorded with the dimensions length-time$^{-1}$, and the conversion factor 0.44704 that expresses units of mph in base units m/s. We will refer to unit objects corresponding to the built-in units as simple units, since each has a dimension part that is a single unit name. A compound unit, such as 10_cm/s, contains a dimension part that is a composite object similar to an algebraic object, comprised of unit names, prefixes, powers, and special *, /, and ^ operators. For either type of unit object, the command UBASE converts an object into an equivalent where the dimension part of the result contains only base units. If the original magnitude is 1, then the result magnitude is the conversion factor from the original units to base units:

$$1 \text{_mph UBASE} \Rightarrow 0.44704 \text{_m/s}$$

You can use the following program to determine the actual dimensions of a unit object:

```
UDIMS UnitDimensions 8A8E

<table>
<thead>
<tr>
<th>UDIMMS Unit Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>level 1</td>
</tr>
<tr>
<td>magnitude_units</td>
</tr>
</tbody>
</table>

<< { } 1 1 1 1 1 1 1 | Initial values.
- list kg m s A K cd mol ? | Make local variables.
<< UBASE OBJ- ->STR | Convert normalized unit object to string.
  4 OVER SIZE SUB | Strip off "1_".
  "\" SWAP + OBJ- | Convert to an expression.
  1 8 | For each base unit:
  FOR i i | Get the i_th base unit.
  { kg m s A K cd mol ? } SWAP GET | Assign it value 10.
  10 OVER STO | Get the power of the base unit.
  OVER EVAL LOG | Restore value 1.
  1 ROT STO | Add the power to the list.
  'list' SWAP STO+ | Discard the expression.
  NEXT | Retrieve the list.
  DROP | Add the unit magnitude at the head.
  list |
  + |
  >> |
  >> |
```

UDIMS converts a unit object into a list, where the first element is the equivalent
magnitude in base units, followed by the object’s dimensions expressed as (integer) powers of length, mass, time, current, temperature, luminous intensity and quantity of matter, in that order. (UDIMS only works for integer dimension powers—see section 21.4.1).

\[ 1_{\text{mph}} \text{ UDIM} S \{ .44104 0 1 -1 0 0 0 0 0 \} \]

### 21.1.3 User-Defined Units
You can represent any unit that can be reduced to SI base units by using compound units. As an additional convenience, you can also create *user-defined units*, to add new unit names that can be used like simple units. A user-defined unit is any global variable that contains a single unit object: the stored object specifies the unit magnitude and dimensions, and the variable name may be used as a unit name in the same manner as any built-in name. Thus

\[ .125_{\text{mi}} \ 'furlong' \ \text{STO} \]

and

\[ 14_{\text{d}} \ 'fortnight' \ \text{STO} \]

create new units furlong and fortnight; then \(1_{\text{furlong/fortnight}}\) is a valid unit object with the dimensions of speed. User-defined units may also be used to define new units, e.g.

\[ 1_{\text{furlong/fortnight}} \ 'slow' \ \text{STO} \]

defines a user-defined unit named slow. Any user-defined unit is subject to the same operations as built-in units:

\[ 1\text{_slow} \ \text{UBASE} \equiv 1.6630952831E-4\text{_m/s} \]

The names of user-defined units should not match any built-in unit, with or without a prefix, as the latter are given precedence in the interpretation of unit objects entered in the command line.

### 21.1.4 Unit Object Mechanics
As for other object types that contain more than one part, the HP 48 provides simple tools for taking unit objects apart, and for assembling them from other objects:
21.1 Unit Management

OBJ→ decomposes a unit object into its magnitude (a real number returned to level 2) and a normalized (magnitude 1) unit object of the same dimensions (level 1).

\[
600 \text{ W/sr OBJ→} 600 1\text{ W/sr}
\]

→UNIT replaces the magnitude of a unit object in level 1 with a real number taken from level 2:

\[
50 100\text{ km/s →UNIT} 50\text{ km/s}
\]

UVAL extracts the magnitude of a unit object:

\[
1.6\times 10^{-19}\text{ C UVAL} 1.6\times 10^{-19}
\]

UVAL and →UNIT are available in the UNITS menu; OBJ→ is found in the menu.

21.2 Unit Conversions

The HP 48’s list of conversion factors for simple units to SI base units makes it straightforward for the calculator to convert any simple or compound unit into any other dimensionally consistent unit. To do so, the HP 48 computes a net conversion factor for the original unit by applying the arithmetic in its dimension part to the conversion factors of the simple units represented there. This value is divided by a net conversion factor similarly computed for the output units, then multiplied by the original unit magnitude. The result is the magnitude of the converted unit.

The most obvious application of this process is in the command CONVERT, which converts one unit (level 2) into an equivalent with its dimension part specified by a second unit object:

\[
1\text{ b*Mpc 1 cm}^3 \text{ CONVERT} 3.08567818585\text{ cm}^3
\]

Note that the magnitude of the second argument unit object is ignored; that argument is only used to specify the new dimensions for the first argument.

If the dimensions of the two units are not consistent, i.e. they do not correspond to the same powers of SI base units, CONVERT returns the Inconsistent Units error. An attempted conversion of a unit object that contains undefined unit names returns the Invalid Unit error.
21.2.1 The Unit Menus

The unit menus are the sixteen built-in menus associated with the \( \text{[\text{UNITS}} \) key. All of the built-in simple units, plus a number of compound units, are represented in these menus, organized by common dimensionality or at least common application. For example, the length menu \( \text{[\text{LENG}} \) contains keys for 22 simple units of length. The speed menu \( \text{[\text{SPEED}} \) has seven entries with the dimensions length/time, plus one, ga, that is a unit of acceleration. (It is more practical to include ga in the speed menu than to have a separate acceleration menu with only one entry). The first three speed menu entries, m/s, cm/s, and ft/s, are compound units. Compound units are included in the units menus as typing aids and for convenient conversions.

The units menus are useful catalogs of the built-in units and their dimensions. Since unit names use upper and lower case letters, but the key label characters are upper-case only, you may occasionally wish to use \( \text{[\text{REVIEW}} \) to see the correct spelling of a unit name. For example, in the second page of the electricity menu \( \text{[\text{ELEC}} \), \( \text{[\text{REVIEW}} \) produces:

![Unit Menus](image)

To determine the value of a particular unit in SI base units, you can use the menu keys like this:

\[
\text{[\text{UNITS}} \quad \text{NXT} \quad \text{ENERG} \quad 1 \quad \text{BTU} \\
\text{[\text{UNITS}} \quad \text{UBASE} \quad 1055.05585262 \text{ kg} \cdot \text{m}^{2} / \text{s}^{2}
\]

Here we entered the unit object 1_BTU by entering 1 then pressing \( \text{[\text{BTU}} \). This behavior is characteristic of units menu keys. Their design arises from the observation that the conversion of a unit object from one unit representation to another has certain similarities to an HP Solve problem. Consider the conversion of square centimeters to square inches. You could achieve such conversions by using a current equation

\[
\text{CM}^2 \times 1_{\text{cm}^2} = 1_{\text{in}^2}
\]
To convert 100 $cm^2$ to $in^2$, you would use the following keystrokes:

`[aptersolve] 100 [units] cm^2 [area] in^2 \Rightarrow \text{IN2: 15.5000310001}.`

Similarly, to convert a number of $in^2$ to $cm^2$, you would store a value for IN2 and solve for CM2.

The HP 48 unit menus are deliberately modeled on the HP Solve system to provide convenient automated conversions among dimensionally consistent units. To achieve the above $cm^2$-to-$in^2$ conversion, you can use these keystrokes:

`[units] area 100 [units] cm^2 [area] in^2 \Rightarrow 15.5000310001\_in^2.`

Notice that the keystrokes after the initial menu selection are identical to the previous example. The result this time is a unit object with the same magnitude as the HP Solve result.

The unit menu behavior demonstrated here is derived from specific menu key actions. An unshifted unit menu key executes the sequence:

1. ENTER (if there is a command line);
2. 1-unit ENTER, where unit is the unit indicated by the menu key label;
3. \( \times \) (multiply).

A left-shifted \( \leftarrow \) menu key works the same way, except that the final multiply is replaced by CONVERT.

Returning to the area conversion example, 100 $cm^2$ enters 100, then multiplies it by 1_cm^2, yielding 100_cm^2. \( \leftarrow \) \( \text{in}^2 \) enters 1_in^2, then executes CONVERT, returning 15.5000310001_in^2.

There are variations on the standard sequence of unit menu keys illustrated above. The multiply performed by the unshifted keys allows you to create compound units by pressing successive menu keys. For example, to convert 2500 dyne-cm to joules, press
A right-shifted (\(\rightarrow\)) unit menu key works like an unshifted key, except that it performs a final / instead of *. Thus, with the previous result still in level 1,

\[
2500 \rightarrow\text{UNITS} \rightarrow\text{NXT} \rightarrow\text{FORCE} \rightarrow\text{DYN}
\]

\[
2500 \rightarrow\text{UNITS} \rightarrow\text{LEN} \rightarrow\text{CM}
\]

\[
2500 \rightarrow\text{UNITS} \rightarrow\text{ENRG} \rightarrow\text{J} \rightarrow\text{.00025}_\text{.J}
\]

A right-shifted (\(\rightarrow\)) unit menu key works like an unshifted key, except that it performs a final / instead of *. Thus, with the previous result still in level 1,

\[
2500 \rightarrow\text{UNITS} \rightarrow\text{TIME} \rightarrow\text{S} \rightarrow\text{.00025}_\text{.J/s}
\]

The unit menu keys literally execute the commands \(\ast\), CONVERT, and \(--\)--if an error occurs, one of these commands is identified in the error message.

The various actions of the unit menu keys described here apply only in immediate-execute mode (section 6.4.1). In algebraic entry mode (ALG) or program entry mode (PRG), an unshifted menu key acts as a simple typing aid for the corresponding unit name (note that this means that to enter a unit object, you must use the \([\rightarrow\text{.J}]\) key as well as the menu key). The shifted keys are inactive.

### 21.2.3 Units in Custom Menus

When a custom menu key (section 7.3) is specified by a unit object in the CST list, the key exhibits the same actions as the menu keys in any of the units menus. This is
particularly useful when you are performing repeated conversions among units that don't happen to be in the same unit menu, or include one or more user-defined units. For example, you can create a dollars/pounds menu as follows, using the user-defined units from the previous section:

```
{ 1_$  1_£ } MENU
```

eyields this display

Here you can use the menu keys for conversions:

```
50  $  £  26.18_£
```

21.3 Unit Object Mathematics

The real power of the HP 48 unit management system comes from the unit conversions that occur automatically when you apply various functions to unit objects. The conversions entail a certain speed penalty compared with the same functions applied to the real number magnitudes alone, but you gain a significant problem-solving efficiency because the calculator does the dimensional bookkeeping for you.

21.3.1 Unit Operations Requiring Dimensional Consistency

A typical unit operation is the addition of two objects with different dimension parts:

```
1_mi  1_km  +  2.609344_km
```

In effect, + applies CONVERT to the two arguments to convert the first argument to the same units as the second, then adds the two magnitudes. The result units are thus always the same as those of the second argument, which is consistent with CONVERT.
If you want a result expressed in the units of the first argument, execute SWAP prior to executing the combining function.

Automatic conversion is performed by all of the following functions:

- + and -.
- %CH, which computes \(100\cdot(x-y)/y\) from two objects \(x\) and \(y\).
- %T. Here the first argument is converted to the dimensions of the second, then the magnitudes are divided. The result is always a real number.
- =. In numerical evaluation mode (section 3.5.5), = is equivalent to −.
- ==, =, >, <, ≤, ≥. These six predicate operators compare the magnitudes of two objects after converting them to common units. Note that SAME does not perform any conversions; two unit objects must have the same magnitudes and dimension parts for SAME to return true. See section 9.3.2.

If one of the two arguments of any of these functions is the real number 0, it is automatically replaced by \(0\_\text{units}\) before the function is applied, where \(\text{units}\) is the dimension part of the non-zero argument. Considered to have the dimensions of the other argument. Thus

\[
1\_m 0 + \Rightarrow 1\_m
\]

Besides being a simple convenience, this feature allows ISOL (section 16.4) to work with expressions containing unit objects. ISOL starts with 0, then accumulates terms as the various inverse functions are applied to the argument expression.

### 21.3.2 Unit Functions with Simplification

Multiplicative functions automatically simplify the dimension part of unit object results so that no unit name appears more than once. Thus

\[
1\_{\text{cm/s}} 1\_{\text{cm}^2/\text{s}} \ast \Rightarrow 1\_{\text{cm}^3/\text{s}^2}
\]

\[
1\_{\text{g/l}} \ast 25\_l \Rightarrow 25\_g.
\]

Unit names in the result’s numerator and denominator are collected, and unit names common to the numerator and denominator are canceled. Note that only identical unit names with common prefixes are collected or canceled. When the net power of any name is zero, the name is removed from the dimension part of the unit. If all of the names in the dimension part cancel, the result is converted to a real number:
Unit simplification is performed by the following functions:

- *, %, /, and INV.
- ^, SQ, √, and XROOT. For ^ and XROOT, the level 1 argument is a real number, which is multiplied or divided into the powers of the unit names in the dimension part. If the argument is zero for XROOT, the Bad Argument Value error is reported (unlike the case where both arguments are real numbers, which returns the underflow exception).

The functions do not attempt to simplify specific combinations of units into equivalent units. 1_N*m, for example, does not automatically convert to 1_J. While this might be convenient in some cases, there is no unambiguous set of rules that the calculator could use to identify desirable combinations. The gasoline mileage of automobiles is an obvious case in point. We commonly express this quantity in miles/gallon or kilometers/liter, but it actually has the dimensions length⁻². It would require a very intelligent conversion algorithm to resist returning mileage in units of m⁻².

When you do want to convert a unit object to a specific form, you can use CONVERT, where the second argument contains the desired dimension part. For cases where you do not know exactly what form is required, but you do want to extract a particular combination unit, use UFACT. This command takes a unit object (level 2) and converts it such that its dimension part becomes the product of the dimension part of a level 1 unit object times whatever base unit factors are left over. Suppose you want to know what a watt is in terms of newtons:

```
1_W 1_N UFACT = 1_N*m/s
```

UFACT is equivalent to the sequence

```
OBJ→ 3 ROLLD / OVER / UBASE *.
```

### 21.3.3 Operations on the Unit Magnitude

The following functions alter the magnitude of a unit object without regard to its dimensions: ABS, CEIL, FLOOR, FP, IP, NEG, RND, SIGN, and TRNC. All of these except SIGN apply a function to the unit magnitude, and combine the result with the original dimension part of the argument. SIGN returns a real number −1, 0, or +1, according to the sign of the magnitude.

RND and TRNC accept a unit object as the level two argument. The level one
argument, which specifies the rounding, must be a real number or a symbolic object.

21.3.4 Trigonometric Unit Functions
The trigonometric functions SIN, COS, and TAN can accept unit object arguments, provided the units are simple angle units—radians (r), degrees (°), arc-minutes (arcmin), arc-seconds (arcsec), or grads (grad). The results are independent of the current angular mode. However, differentiation and integration of expressions containing these functions with unit object arguments will produce correct results only in radians mode. Note also that there are no corresponding inverse functions that return unit objects.

21.3.5 Examples of Calculations with Units

Example. A mass of 15 g moves in a circle of radius 7 cm with a speed of 35 cm/s. Compute its various kinematic and dynamic properties.

Solution. Start by storing the parameters:

15_g 'M STO 7_cm 'R STO 35_cm/s 'V STO

Now compute the angular velocity:

\[ \omega = \frac{V}{R} = \frac{35}{7} \text{ cm/s} \]

\[ \omega \text{ STO} \]

Moment of inertia:

\[ I = M \times R^2 \]

\[ I \text{ STO} \]

Angular momentum:

\[ L = I \times \omega \]

\[ L \text{ STO} \]

Kinetic Energy:

\[ KE = \frac{1}{2} I \times \omega^2 \]

\[ 1\_J \text{ CONVERT} \]

Example. How large must a parallel-plate air-filled capacitor be to have a capacitance
of 1 farad, if the plate separation is 0.1 mm?

**Solution.** The capacitance of a parallel-plate capacitor is given by \( C = \varepsilon_0 A/d \), where \( A \) is the plate area, \( d \) is the separation, and \( \varepsilon_0 = 8.9 \times 10^{-12} \text{coul}^2/\text{N} \cdot \text{m}^2 \). Rearranging, \( A = dC/\varepsilon_0 \). Thus \( A \) can be calculated by

\[
.1 \text{ mm} \quad 1 \text{ F} \quad * \quad 8.9E-12 \text{ C}^2/(\text{N} \cdot \text{m}^2) \quad / \quad \text{UBASE} \quad \Rightarrow \quad 1.1E7 \text{ m}^2.
\]

21.3.6 HP Solve

The current equation used by HP Solve and any of its variables may contain unit objects, either explicitly in expressions or implicitly as the values of variables within those expressions. The only constraint is that you must supply a guess for the unknown variable which is a unit object with the correct dimensions for that variable. See section 14.3 for more detail and examples.

When the root-finder evaluates the current equation at each iteration, it evaluates it in the normal way, then strips the units from the result for comparison with the previous iteration. If the current equation happens to change its units at some point in its domain, this would appear as a discontinuity to the root-finder, leading possibly to an invalid result. This situation arises from terms like this:

\[
\text{IFTE}(x<0, 1 \text{ cm}, 10 \text{ mm}).
\]

Such terms are unlikely to appear in practical problems.

21.3.7 Plotting

The automatic plotting provided by DRAW (Chapter 15) does not fully support units, in that the plotting ranges may only be specified as simple real numbers. However, you can still create function plots from current equations that contain unit objects, with the following provisions:

- You must store a unit object with appropriate dimensions in the independent variable. The magnitude of the object is not used, but the dimension part determines the units of the independent variable at each evaluation of the current equation as the plot progresses across the horizontal domain.

- The horizontal and vertical ranges are taken to be the magnitudes of unit objects. That is, if the current equation evaluates to \( y_{\text{units}_y} \), then the vertical range is assumed to be \( y_{\text{min\_units}_y} \) to \( y_{\text{max\_units}_y} \), where \( y_{\text{min}} \) and \( y_{\text{max}} \) are the (dimensionless) vertical plot limits. The horizontal range is effectively \( x_{\text{min\_units}_x} \) to \( x_{\text{max\_units}_x} \), where \( \text{units}_x \) is the dimension part of the unit object stored in the
independent variable.

- The left and right sides of an equation should evaluate to the same units. Otherwise they will be plotted with different vertical scales. \texttt{ROOT} and \texttt{ISECT} will still give valid results, but the graph will show the intersections in the wrong places.

As in the case of HP Solve, an artificial discontinuity can be introduced if the current equation for plotting returns different units at different points in the plot domain.

\textbf{21.3.8 Differentiation}

Consider the simple conservation of energy for a falling body: $\frac{1}{2}v^2 = gh$, where $v$ is the body's speed after it falls a distance $h$. Entering this equation into the HP 48, you can use \texttt{QUAD} (section 16.4.3) to solve for $V$:

\[
'\frac{V^2}{2} = gh' \quad 'V' \quad \text{QUAD} \quad 1 \quad 's1' \quad \text{STO} \quad \text{EVAL} \quad \text{COLCT} \quad \therefore \quad 'V = \sqrt{2gh}'.
\]

Then you can store values for $g$ and $h$, and evaluate again:

\begin{verbatim}
1 \text{FIX} 9.81 \text{m/s}^2 \quad 'g' \quad \text{STO} \quad 100 \text{m} \quad 'h' \quad \text{STO} \quad \text{EVAL} \quad \text{UBASE} \quad \therefore \quad 44.3 \text{m/s}.
\end{verbatim}

But if you repeat the exercise, this time with the unit objects already stored, you get an unexpected error:

\[
'\frac{V^2}{2} = gh' \quad 'V' \quad \text{QUAD} \quad \therefore
\]

The problem arises from the differentiation performed by \texttt{QUAD} as it computes the quadratic coefficients. During the course of chain-rule differentiation, an expression is evaluated one or more times, which means that variable names are replaced by their values. Try differentiating the right-hand side of the equation with respect to $V$:

\[
'\frac{\partial V(g \cdot h)}{\partial V} \quad \text{EVAL} \quad \therefore \quad '\frac{\partial V(g \cdot h)}{\partial V} + g \cdot \frac{\partial V(g)}{\partial V}'.
\]
At the next evaluation, both of the remaining derivatives are zero, but the values of \( h \) and \( g \) are also substituted in the expression, so that you will be evaluating \('0*100_m+9.81_m/s^2*0'\). This fails because the two terms have different unit dimensions.

This problem affects \( \partial \), QUAD, and TAYLR applied to expressions with variables containing unit objects. There are two ways to avoid the difficulty:

- Carry out the differentiations before assigning values to the variables, or switch to a directory where the variables are undefined.
- Replace the variable names by their values before differentiating. For the variable of differentiation, separate its units from the variable.

To illustrate the second method, start again by entering the equation \('V^2/2=g*h'\). First, evaluate \( g \) and \( h \) using SHOW (section 16.4.2):

\[
\text{SHOW} \quad \text{'}V^2/2=g*h' \quad \{ \text{V} \} \quad \Rightarrow \quad \text{'}V^2/2=981_m^2/s^2' 
\]

Now replace \( V \) by \('v*1_m/s'\):

\[
\{ \text{V} \quad \text{'}v*1_m/s' \} \quad \text{MATCH DROP} \quad \Rightarrow \quad \text{'}(v*1_m/s)^2/2=981_m^2/s^2' 
\]

Solve for \( v \):

\[
\text{QUAD EVAL} \quad \Rightarrow \quad \text{'}v=44.3' 
\]

You can multiply this result by \( 1_m/s \) to obtain \( V \).

### 21.3.9 Integration

For numerical evaluation of an integral, both the integrand and the integration limits may contain unit objects. There is nothing special that you need to do to compute an integral with units, except to insure that the units of the limits are compatible with the units assumed for the independent variable:

- The units of the lower limit, determined by evaluating it numerically, are used for the integration variable at every point at which the integrand is evaluated during the approximation process. The limit's units must therefore have suitable dimensions for successful numerical evaluation of the integrand.
- The units of the upper limit must be dimensionally consistent with those of the lower limit.
• The first time the integrand is evaluated, the units of the integrand are saved. Subsequent evaluations of the integrand are converted to the dimensions of the saved units. This prevents any variation of units at different points in the integral domain from causing artificial discontinuities (integration is thus a little "smarter" than DRAW or HP Solve).

• The units of the result are the product of the integrand units and the units of the limits.

Example. Compute \( \int_{0}^{10} (10\text{ cm/s} + 5\text{ cm/s}^2 \cdot t) \, dt \).

Solution.
\[
\left( \int_{0}^{10\text{ s}} (10\text{ cm/s} + 5\text{ cm/s}^2 \cdot t) \, dt \right) \rightarrow \text{NUM} \quad 350\text{ cm}
\]

Symbolic integration with units presents a somewhat different problem than numerical integration. There are integrals that appear to be computable but which return an error on the HP 48. Consider the integral
\[
\int_{a}^{b} \frac{1}{k \cdot x + v} \, dx
\]
where \( k \) has the dimensions \( s^{-1} \), \( a \), \( b \) and \( x \) are expressed in \( m \), and \( v \) is a speed in \( m/s \). Evaluating this on the HP 48 would return an error, because \( ln \) requires a dimensionless argument. To compute the integral correctly, you must factor the units out of the denominator before integrating.

21.4 Unit Management Idiosyncrasies

The HP 48 unit management system is not perfect, in that in a few areas it compromises universality for the sake of execution speed or simplicity. This may lead to some surprises, but there are actually very few practical calculations that you can not carry out as long as you allow for the system's limitations.

21.4.1 Non-integer Unit Powers

The units in the dimension part of a unit object may be raised to arbitrary non-integer powers. CONVERT computes its conversion factor accordingly, so that you can convert between units with non-integer dimensions:

\[
1\text{ m}^{.5} \quad 1\text{ cm}^{.5} \quad \text{CONVERT} \quad 10\text{ cm}^{.5}
\]
However, for the sake of dimension checking, unit powers are rounded to integers in the range \(-128\) to \(+127\). This means that while you can successfully add \(1\text{_m}^{\cdot5}\) and \(1\text{_cm}^{\cdot5}\), you can also add \(1\text{_m}^{\cdot5}\) and \(1\text{_cm}^{\cdot6}\) with no error report. The dimension checking limitation also carries over to UBASE, and UFACT. Neither of these commands will return meaningful results for units containing non-integer exponents.

This limitation is a deliberate design choice intended to make unit object arithmetic as fast as possible. Representing base unit powers with small integers provides a considerable efficiency compared with using floating-point numbers. The loss of ability to handle non-integer exponents in all circumstances is presumed to be a modest price to pay for the speed gain, given that the great majority of physical problems involves integer powers of units.

There are some cases where non-integer exponents appear in the course of a calculation even though the final result has only integer powers. You can perform such calculations on the HP 48, as long as you take care to simplify quantities to base units whenever possible, before any conversions are done. For example, evaluating

\[
2\text{_atm} \ 8\text{_lb/gal} \ * \ \sqrt{\text{UBASE}} \ = \ 13937.812022\text{_kg}^2/(\text{m}^2\text{s}),
\]

which is dimensionally incorrect. The UBASE should be applied before the square root to obtain the correct result:

\[
2\text{_atm} \ 8\text{_lb/gal} \ * \ \sqrt{\text{UBASE}} \ = \ 13937.812022\text{_kg}/(\text{m}^2\text{s}).
\]

### 21.4.2 Temperature

The temperature units °F and °C have ambiguous meanings that prevent them from fitting easily in the unit management system. They can be interpreted as units of thermodynamic temperature; with that definition, °F and °C are interchangeable with °R and K, respectively. Conversions associated with this interpretation work properly, such as converting J/K to BTU/°F.

An alternate meaning for these units is as points on a thermometer scale. These are not units in the usual sense, and intermixing thermometer points with thermodynamic temperature units leads to problems. The HP 48 unit conversion scheme is based on multiplicative conversions. This means that a conversion between two dimensionally consistent units can be achieved with a simple multiplication. °F and °C considered as thermometer points are not expressible as simple multiples of the base unit K. Additive constants are needed as well, so the ordinary conversion scheme is inadequate for these units. The thermometer points do not, for example, obey ordinary arithmetic rules. For
example, multiplication is not very meaningful--returning 20°F as the value of 2×10°F is not a very useful result. Also, addition is not commutative. If you write 5°C+10°F, you probably mean “increment a thermometer reading of 5°C by 10°F (i.e. 5/9×10°C).” But this does not yield the same result as 10°F+5°C.

The HP 48 does make some provision for the use of °F and °C as thermometer points. In particular, when CONVERT is applied to two unit objects whose dimension parts are each any one of the four temperature units, the conversion accounts for the additive constants. This allows CONVERT, including the automatic CONVERT built into the TEMP menu keys, to be used for simple temperature conversions:

\[
98.6_°F \ 1_°C \ \text{CONVERT} \ {\Rightarrow} \ 37_°C.
\]

The HP 48 computes this result by converting the first unit object to kelvins, then converting back to the units of the second object, including additive constants in both conversions. For better or worse, a similar logic is applied to addition and subtraction. Thus we have the somewhat surprising result

\[
10_°C \ 5_°C \ + \ {\Rightarrow} \ 288.15_°C.
\]

Both arguments are converted to kelvins, added, then converted back to °C. This result is justifiable, but it is hardly useful. The consolation is that the physical laws used in practical problems are generally written in terms of absolute temperatures, and the HP 48 units K and °R always work in calculations related to these laws.

A safe strategy when you do want to enter temperatures in °C or °F is to apply the UBASE function to the temperatures prior to any further calculations. For example, in problems of thermal expansion, you might have an expression of the form

\[
\text{Length}=L_0*(1 + \alpha *(T-T_0)),
\]

where L₀ is the length of a bar at the reference temperature T₀, and α is the coefficient of expansion. Calculations with this formula will not give correct results if the temperatures are entered in °C or °F, unless it is modified to

\[
\text{Length}=L_0*(1 + \alpha *(\text{UBASE}(T)-\text{UBASE}(T_0))).
\]

Now the temperatures may be entered in any units. α may also be expressed in units K⁻¹, °R⁻¹, °F⁻¹, or °C⁻¹. Note that applying UBASE to the temperature difference, e.g. UBASE(T-T₀), does not solve the problem, because the subtraction is performed before the conversion to base units.
21.4.3 Angle Units

The angle units in the ANGL menu are dimensionless, which presents another problem for the unit management system. Consider the radians unit \( \pi \). A typical use of radians is in formulae such as the definition of angular speed, \( \omega = \frac{v}{R} \). Here \( v \) has dimensions \( \text{length} \cdot \text{time}^{-1} \) and \( R \) has dimension \( \text{length} \), and so \( \omega \) must have the dimension \( \text{time}^{-1} \). But instead we like to assign \( \omega \) the dimensions \( \text{radians} \cdot \text{time}^{-1} \) to tie it to angular motion. Evidently, \( \text{UBASE}(1_\pi) \) must have the value 1 (dimensionless) to make calculations like \( \omega = \frac{v}{R} \) return correct results. But if this is the definition of \( 1_\pi \), then \( 1_\pi/s \) would convert to \( 1_\text{Hz} \), which is certainly wrong by a factor of \( 1/2\pi \). In order to make this conversion correct, \( \text{UBASE}(1_\pi) \) must have the value \( 1/2\pi \).

This contradiction is insoluble within the scope of the HP 48 unit management system, which can not deal consistently with dimensionless units that appear or disappear depending on what the visual appearance of an expression is supposed to be. The HP 48 makes the somewhat arbitrary choice that \( \text{UBASE}(1_\pi) \) is \( 1/2\pi \) (actually \( .159154943092 \)). With analogous definitions for \( 1_\circ \) (1/360), \( 1_\text{grad} \) (1/400), \( 1_\text{arcmin} \) (1/21600), and \( 1_\text{arcsec} \) (1/1296000), conversions among all these units and Hz give correct results.

To deal with formulae like the angular speed example where angle units can appear and disappear, you can construct expressions in which each term must explicitly or implicitly be uniform in angle units—specifically, each should have the same power of angle units. For example, if the angular frequency equation is represented by the object \( \omega = \left(\frac{v}{R}\right) * 1_\pi \), you can use it with values of \( \omega \) expressed in \( \pi \), Hz, or any other units \( \text{angle/time} \) (but you can’t use units of \( 1/\text{time} \)). Alternatively, you can leave the angular units out entirely, since they do not contribute anything to the computation.

By extension from the plane angle case, and for the benefit of conversions involving photometric units (see the next section), the solid angle unit \( \text{sr} \) is defined such that \( \text{UBASE}(1_{\text{sr}}) \) returns \( 1/4\pi \).

21.4.4 Photometric Units

The photometric units included in the LIGHT menu provide various measures of luminous power. The term luminous refers to the incorporation of the spectral response of the idealized human eye into the units, so that they are not directly convertible into ordinary units of power. For example, luminous intensity is the luminous power emitted per steradian by a body. The base unit of luminous intensity is the candela, abbreviated cd. It is defined as the luminous intensity of a body emitting \( 1/683 \) watt/sr of monochromatic radiation at \( 5.4 \times 10^{12} \text{ Hz} \) (5550 angstrom), which is the wavelength of maximum sensitivity of the eye). Because of the spectral response factor, it takes more power at any other wavelength to produce the same luminous intensity.
The fact that the definition of candela includes solid angle makes conversions among related units problematic for the HP 48 unit management system, again because of the elusive nature of angle units. For example, a lumen (lm) is a measure of luminous flux, which is the power emitted by a surface per unit solid angle. A lumen is defined as the flux from 1 cd into 1 sr. However, if you execute 1_lm 1_cd CONVERT, you will obtain 7.95774715459E-2_cd. The numerical magnitude of the result is 1/4π; this factor is included to cancel the 1/4π that is included in the definition of the HP 48 sr unit. That is, a more proper conversion between lumens and candelas should include an explicit solid angle factor:

\[ 1_{lm}/sr \ 1_{cd} \ \text{CONVERT} \rightarrow 1_{cd}. \]

A conversion between cd and lm without including solid angle is essentially invalid, but the HP 48 cannot detect this because solid angle is dimensionless.

The LIGHT menu also includes three units of luminous surface brightness--stilb (sb), lambert (lam), and footlambert (flam), and three units of illuminance--phot (phot), lux (lux), and footcandle (fc). Surface brightness is the power emitted by a unit surface area per unit solid angle, whereas illuminance is the power received by a unit surface area. Dimensionally, the two quantities are related to each other in the same manner as luminous intensity and luminous flux. Therefore conversions between the two types of units must include a solid angle factor, either multiplied times the surface brightness unit, or divided into the illuminance unit:

\[ 1_{stilb}*sr \ 1_{ph} \ \text{CONVERT} \rightarrow 1_{ph}. \]
## Program Index

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD&amp;T</td>
<td>Add Date and Time</td>
<td>627</td>
</tr>
<tr>
<td>ADDV</td>
<td>Concatenate Vectors</td>
<td>300</td>
</tr>
<tr>
<td>AGXOR</td>
<td>Animate with GXOR</td>
<td>264</td>
</tr>
<tr>
<td>APLY1</td>
<td>Apply Program to 1 Symbolic Array</td>
<td>304</td>
</tr>
<tr>
<td>APLY2</td>
<td>Apply Program to 2 Symbolic Arrays</td>
<td>304</td>
</tr>
<tr>
<td>APVIEW</td>
<td>Animation with PVIEW</td>
<td>268</td>
</tr>
<tr>
<td>AREPL</td>
<td>Animation with REPL</td>
<td>268</td>
</tr>
<tr>
<td>ASN41</td>
<td>ASN HP 41-style</td>
<td>182</td>
</tr>
<tr>
<td>ASTO</td>
<td>Animation with STO</td>
<td>268</td>
</tr>
<tr>
<td>BINCALC</td>
<td>Binary Integer Calculator</td>
<td>192</td>
</tr>
<tr>
<td>BOUNCE</td>
<td>Bouncing Ball Demo</td>
<td>269</td>
</tr>
<tr>
<td>BS?</td>
<td>Bit Set?</td>
<td>178</td>
</tr>
<tr>
<td>C1ADD1</td>
<td>Column 1 Added with 1's</td>
<td>661</td>
</tr>
<tr>
<td>C1SPLIT</td>
<td>Column 1 Split</td>
<td>660</td>
</tr>
<tr>
<td>CB</td>
<td>Clear Bit</td>
<td>178</td>
</tr>
<tr>
<td>CDIST</td>
<td>$x^2$ Probability Density</td>
<td>677</td>
</tr>
<tr>
<td>CEQN</td>
<td>Characteristic Equation</td>
<td>310</td>
</tr>
<tr>
<td>CHARDISP</td>
<td>Display HP 48 Characters</td>
<td>258</td>
</tr>
<tr>
<td>CHIMES</td>
<td>Hour Chimes</td>
<td>636</td>
</tr>
<tr>
<td>CHKINPUT</td>
<td>Prompt and Check Input</td>
<td>331</td>
</tr>
<tr>
<td>CHVAR</td>
<td>Change of Variables</td>
<td>614</td>
</tr>
<tr>
<td>CI</td>
<td>Cosine Integral</td>
<td>369</td>
</tr>
<tr>
<td>CINT</td>
<td>Circle in a Triangle</td>
<td>248</td>
</tr>
<tr>
<td>CLADD1</td>
<td>Column Last Added with 1's</td>
<td>661</td>
</tr>
<tr>
<td>CLSPLIT</td>
<td>Column Last Split</td>
<td>661</td>
</tr>
<tr>
<td>COSSUM</td>
<td>Cosine of a Sum</td>
<td>573</td>
</tr>
<tr>
<td>COUNT4</td>
<td>Count in 4 Ranges</td>
<td>229</td>
</tr>
<tr>
<td>CROSSF</td>
<td>CROSS Function</td>
<td>289</td>
</tr>
<tr>
<td>CSC</td>
<td>Cosecant Function</td>
<td>537</td>
</tr>
<tr>
<td>DATENAME</td>
<td>Create a Name from the Current Date</td>
<td>107</td>
</tr>
<tr>
<td>DD&amp;T</td>
<td>Delta Days and Time</td>
<td>627</td>
</tr>
<tr>
<td>DELCHIMES</td>
<td>Delete Chimes Alarm</td>
<td>638</td>
</tr>
<tr>
<td>DELOLDALARMS</td>
<td>Delete Old Alarms</td>
<td>638</td>
</tr>
<tr>
<td>DELROW</td>
<td>Delete a Matrix Row</td>
<td>280</td>
</tr>
<tr>
<td>DFACT</td>
<td>Double Factorial</td>
<td>234</td>
</tr>
<tr>
<td>DIM</td>
<td>Symbolic Array Dimensions</td>
<td>302</td>
</tr>
<tr>
<td>DOTT</td>
<td>DOT Function</td>
<td>288</td>
</tr>
<tr>
<td>DOW</td>
<td>Day of Week</td>
<td>625</td>
</tr>
<tr>
<td>DRAWEBAR</td>
<td>Draw with Error Bars</td>
<td>655</td>
</tr>
<tr>
<td>DRAWPX</td>
<td>DRAW using PIXON</td>
<td>271</td>
</tr>
<tr>
<td>FDER</td>
<td>Formal Derivative</td>
<td>593</td>
</tr>
<tr>
<td>FDIST</td>
<td>F-Distribution Probability Density</td>
<td>677</td>
</tr>
<tr>
<td>FIB</td>
<td>Fibonacci Series Generator</td>
<td>296</td>
</tr>
<tr>
<td>FIND</td>
<td>Find a Variable</td>
<td>121</td>
</tr>
<tr>
<td>FRACALC</td>
<td>Fraction Calculator</td>
<td>194</td>
</tr>
<tr>
<td>FRAME</td>
<td>Frame the Graph Screen</td>
<td>273</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>FUNOF?</td>
<td>607</td>
<td></td>
</tr>
<tr>
<td>GCD</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>GSAMP</td>
<td>261</td>
<td></td>
</tr>
<tr>
<td>GSORT</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>INDEF</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>INFSUM</td>
<td>349</td>
<td></td>
</tr>
<tr>
<td>INTEVAL</td>
<td>603</td>
<td></td>
</tr>
<tr>
<td>INTSAMP</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>ITDRAW</td>
<td>503</td>
<td></td>
</tr>
<tr>
<td>JULIA</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>KEEP</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>KEYHALT</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>KEYTIME</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>LCM&amp;GCD</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td>LINEAR?</td>
<td>606</td>
<td></td>
</tr>
<tr>
<td>LPWR</td>
<td>584</td>
<td></td>
</tr>
<tr>
<td>LSFIT</td>
<td>658</td>
<td></td>
</tr>
<tr>
<td>MAND.EQ</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>MANPLOT</td>
<td>481</td>
<td></td>
</tr>
<tr>
<td>MEDIAN</td>
<td>646</td>
<td></td>
</tr>
<tr>
<td>MINFSUM</td>
<td>349</td>
<td></td>
</tr>
<tr>
<td>MINISTK</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>MINL</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>MINOR</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>MNDROP</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>MNORM</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>MSGSHOW</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>NDIST</td>
<td>677</td>
<td></td>
</tr>
<tr>
<td>NEXTDAY</td>
<td>634</td>
<td></td>
</tr>
<tr>
<td>NLFIT</td>
<td>666</td>
<td></td>
</tr>
<tr>
<td>NORM</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>NSEGINT</td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>N-S</td>
<td>303</td>
<td></td>
</tr>
<tr>
<td>OLABEL</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>PADD</td>
<td>580</td>
<td></td>
</tr>
<tr>
<td>PDIVD</td>
<td>582</td>
<td></td>
</tr>
<tr>
<td>PEXPR</td>
<td>582</td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>584</td>
<td></td>
</tr>
<tr>
<td>PMUL</td>
<td>581</td>
<td></td>
</tr>
<tr>
<td>PNEG</td>
<td>581</td>
<td></td>
</tr>
<tr>
<td>POIS</td>
<td>356</td>
<td></td>
</tr>
<tr>
<td>POLY</td>
<td>585</td>
<td></td>
</tr>
<tr>
<td>POLYF</td>
<td>536</td>
<td></td>
</tr>
<tr>
<td>POLYFIT</td>
<td>664</td>
<td></td>
</tr>
<tr>
<td>PPWR</td>
<td>583</td>
<td></td>
</tr>
<tr>
<td>PREST</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>PRGINT</td>
<td>615</td>
<td></td>
</tr>
<tr>
<td>PRIMES</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>PROMPTCONT</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>PSAVE</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>PSUB</td>
<td>580</td>
<td></td>
</tr>
<tr>
<td>Program Index</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>PTINFSUM</td>
<td>Infinite Sum from Previous Term</td>
<td>368</td>
</tr>
<tr>
<td>QU</td>
<td>Quadratic Root Finder</td>
<td>319, 326</td>
</tr>
<tr>
<td>RC-R</td>
<td>Real/Complex to Real</td>
<td>227</td>
</tr>
<tr>
<td>RMINL</td>
<td>Recursive Minimum of a List</td>
<td>353</td>
</tr>
<tr>
<td>→RPN</td>
<td>Convert to RPN</td>
<td>534</td>
</tr>
<tr>
<td>SADD</td>
<td>Add Symbolic Arrays</td>
<td>305</td>
</tr>
<tr>
<td>→SA</td>
<td>Stack to Symbolic Array</td>
<td>302</td>
</tr>
<tr>
<td>SA~</td>
<td>Symbolic Array to Stack</td>
<td>302</td>
</tr>
<tr>
<td>SB</td>
<td>Set Bit</td>
<td>177</td>
</tr>
<tr>
<td>SCDRAW</td>
<td>Save Coordinates and Draw</td>
<td>482</td>
</tr>
<tr>
<td>SCOF</td>
<td>(Unsigned) Symbolic Cofactor</td>
<td>308</td>
</tr>
<tr>
<td>SDET</td>
<td>Symbolic Determinant of a Matrix</td>
<td>308</td>
</tr>
<tr>
<td>SEGINT</td>
<td>Segment an Integral</td>
<td>612</td>
</tr>
<tr>
<td>SETCHIMES</td>
<td>Set Chime Alarm</td>
<td>637</td>
</tr>
<tr>
<td>SETTICKS</td>
<td>Store Tick Spacings</td>
<td>463</td>
</tr>
<tr>
<td>SI</td>
<td>Sine Integral</td>
<td>369</td>
</tr>
<tr>
<td>SIMEQ</td>
<td>Simultaneous Equations</td>
<td>364</td>
</tr>
<tr>
<td>SKETCH</td>
<td>Sketch Lines</td>
<td>273</td>
</tr>
<tr>
<td>SLVDRAW</td>
<td>Solve and Draw</td>
<td>483</td>
</tr>
<tr>
<td>SMINOR</td>
<td>Minor of a Symbolic Matrix</td>
<td>309</td>
</tr>
<tr>
<td>SMS</td>
<td>Scalar Multiply Symbolic Arrays</td>
<td>306</td>
</tr>
<tr>
<td>SMUL</td>
<td>Multiply Symbolic Arrays</td>
<td>306</td>
</tr>
<tr>
<td>SORT</td>
<td>Sort a List in Increasing Order</td>
<td>297</td>
</tr>
<tr>
<td>ΣPATTERNS</td>
<td>Summation Patterns</td>
<td>597</td>
</tr>
<tr>
<td>SSUB</td>
<td>Subtract Symbolic Arrays</td>
<td>305</td>
</tr>
<tr>
<td>STAR</td>
<td>Draw a Star</td>
<td>272</td>
</tr>
<tr>
<td>STRN</td>
<td>Transpose Symbolic Array</td>
<td>305</td>
</tr>
<tr>
<td>SUBCOL</td>
<td>Subtract Columns</td>
<td>282</td>
</tr>
<tr>
<td>SUM4</td>
<td>Sum 1/4</td>
<td>347</td>
</tr>
<tr>
<td>SUMTERM</td>
<td>Compute an Infinite Sum from TERM</td>
<td>347</td>
</tr>
<tr>
<td>S→N</td>
<td>Symbolic to Numeric</td>
<td>303</td>
</tr>
<tr>
<td>TAYLRX0</td>
<td>Taylor's Polynomial at x₀</td>
<td>595</td>
</tr>
<tr>
<td>TDIST</td>
<td>t Probability Density</td>
<td>677</td>
</tr>
<tr>
<td>TICKAXES</td>
<td>Draw Axes with Ticks</td>
<td>464</td>
</tr>
<tr>
<td>TIMED</td>
<td>Timed Execution</td>
<td>350</td>
</tr>
<tr>
<td>TOCOSUM</td>
<td>To the Cosine of a Sum</td>
<td>574</td>
</tr>
<tr>
<td>TPix</td>
<td>Toggle a Pixel</td>
<td>272</td>
</tr>
<tr>
<td>UDIMS</td>
<td>Unit Dimensions</td>
<td>682</td>
</tr>
<tr>
<td>VANGLE</td>
<td>Angle Between Two Vectors</td>
<td>283</td>
</tr>
<tr>
<td>VARPAR</td>
<td>Variable 8t Parametric DRAW</td>
<td>500</td>
</tr>
<tr>
<td>VARPOL</td>
<td>Variable 8θ Polar DRAW</td>
<td>496</td>
</tr>
<tr>
<td>VSUM</td>
<td>Sum Vector Elements</td>
<td>234</td>
</tr>
<tr>
<td>XARCHIVE</td>
<td>Extended Archive</td>
<td>108</td>
</tr>
<tr>
<td>XFORM</td>
<td>Coordinate Transformation</td>
<td>289</td>
</tr>
<tr>
<td>XINT</td>
<td>Extended Integration</td>
<td>607</td>
</tr>
<tr>
<td>XJULIA</td>
<td>Julia Set Utility with Exit</td>
<td>506</td>
</tr>
<tr>
<td>XPTINFSUM</td>
<td>Infinite Sum in x from Previous Term</td>
<td>368</td>
</tr>
<tr>
<td>XSSUM</td>
<td>Extended Sum</td>
<td>597</td>
</tr>
</tbody>
</table>
Subject Index

aborting programs 316
ABS 283
ACK 632, 640
ACKALL 632, 640
acknowledge alarm 179, 631
action 30, 32
user key 179
activation 30
add fractions 569
alarm 627
acknowledge 179, 631
announcement 630
appointment 630
beep 179
catalog 135, 628, 638
commands 636
test 30, 32
control 635
deletion 637, 640
execution object 628, 629, 636
index 635, 636
list 627, 636, 639
parameters 628
repeat interval 636
alarm-set menu 628
ALG annunciator 141
alert annunciator 631
algebraic 48, 515
calculator 21
test mode 141, 143, 207
evaluation 49
object 3, 21, 25, 29, 30, 32, 47, 63, 200, 515, 534
test 16, 48
algebraic/program mode 142
ALOG 570
alpha key action 179
ALRMDAT 630, 632, 640
analytic function 26, 527
angle mode 175, 178, 286
annunciator, alert 631
ALG 141
busy 145, 631
PRG 141
USER 180
1USR 180
applications 402
APPLY 536
appointment alarms 627, 640, 629, 630, 636
arbitrary integer 530, 531, 532
sign 530, 531, 532
ARC 273, 274, 475
ARCHIVE 106
AREA 487
argument 16
disappearing 79
recovery 72, 87, 179, 242, 312
saving 87
array 40, 277, 301
test 151
→ARRY 72, 277, 288
ARRY→ 278
ASCII files 215
ASN 180, 182, 184
assignment, key 175, 180
associate left 565
right 565
association 546, 547, 562, 565
attach library 101
ATTACH 102, 104
ATTN 133, 241, 315, 339
automate linefeed 179
mode change 142
repeated operations 561
simplification 52, 57
automate calculations 195
AUTO 449, 453, 508, 510, 512
AXES 454, 462, 493
→A 565
A→ 565
backspace 168
backup object 46
Bad Argument Type 49
Bad Argument Value 690
Bad Guess(es) 428
BARPLOT 510
BAR 464, 509
base 165
base units 680
BASIC 2, 32, 201, 245, 516
beep, error 179
BEEP 630
best fit 653
BESTFIT 651, 653
BIN 178
binary integer 40
binary transfer 179
binomial coefficients 670
Subject Index

BINS 512, 644
BLANK 262
BOX 273, 274, 475
branch 225
  unconditional 225
built-in object 59
  program object 31
  units 682
busy annunciator 145
BYTES 10, 41, 224, 318, 327, 406
  calculus 587
CASE structure 228
catalog, alarm 638
  equation 421, 638
  statistics 642
cell 150
  cursor 150
CENTR 138, 457, 469
CENT 454, 469
CF 55, 176, 222
%CH 590, 689
chain-rule 587
change of sign 430
changing variable contents 117
character code 38, 39
characteristic equation 310
checksum 327
CHR 39
CIRCL 475
Circular Reference 87
clear flag 175
CLEAR 68, 135
clearing 68
clipping 275
CLKADJ 624
CLLCD 257, 258
clock 621
  setting 622
closing subexpression 160
CLUSR 96
CLVAR 96, 142, 420
CLΣ 642
code object 30
cofactor 307
COLCT 519, 525, 544, 545, 559, 572, 576, 579, 601, 604
column number 116
column vector 282
COLE 507
combinations 669
combining RPN and algebraic 22
COMB 669, 670
comes due 628
command 26, 59, 100
  test 221, 223
command line 63, 139, 140, 144, 146, 202, 335
command stack 145
comment 146, 36, 216
common notation 17
commutation 564
commutative property 562
compact format 9, 405
complex array, MatrixWriter 151
complex number 34
  operators 568
  result 35
composite object 30, 51, 63, 289, 299
compound unit 682
CON 117, 120, 279, 294
concatenation 37
conditional 32, 221, 225
configuration program, library 103
conic plots 459, 488
CONIC 464, 489
CONJ 568
conservative approach 519
constant, symbolic 57, 170, 178, 211
Constant? 429
CONT 183, 290, 314, 316, 329, 333
continuous plot 449, 459
  type 449
contravariant vector 282
control alarms 627, 640, 629, 635
conversions, unit 684
CONVERT 684, 686, 687, 690
coordinate mode 285
coordinate system 178
coordinates, logical 269, 456
  cylindrical polar 284-286
  polar 34, 284-286
  rectangular 284
  spherical polar 284-286
copying stack objects 70
correlation coefficient 648
CORR 648
counted string 37
counter 119
covariant vector 282
COV 648
CORDIR 89
CROSS 283, 288
CST 185, 687
  menu 185, 410
current directory 89, 111, 608
Subject Index

equation 414, 453
path 89, 101, 592
statistics matrix 642
cursor 457
cell 150
graphic 179
subexpression 170
curve filling 179, 499
flag 499
custom error 242
custom menu 185, 186, 175, 443, 687
customization 175
cylindrical polar coordinates 284, 286
C-PX 270, 274
C-R 35
-D 566
data object 32
date display 622
format 179, 623
flag 623
numbers 622
DATE+ 626
DATE 624, 626
→DATE 624
DBUG 317, 319
DDAYS 626
debugging 317
DEC 178
decimal digits 179
DECR 117, 119
default guess 433
DEFINE 86, 205, 208, 210, 526
defining expression 206
procedure 11, 246, 407, 538
definite integral 600
definite loop 230, 234
definition equation 478
definition expansion 570
definition, object 27
→DEF 570
degrees mode 589
degrees of freedom 676
→DEL 147
DEALARM 635, 637
deleting suspended program 316
delimiter 27, 127, 144, 214
DEKEYS 183
DEL→ 147
denominator 163
dependent column number 647
variable 416, 646
depn 454
DEPN 460, 462, 489, 502
DEPTH 293
der 590
derivative 436, 538, 587
DETACH 104
determinant 307
differentiation 436, 538, 587
with units 693
digit-group commas 34
digitize 430
dimensions 679
consistency 688
directory 45, 60, 88, 420
current 89, 111
PURGE 94
Directory Not Allowed 94, 95
Directory Recursion 95
disappearing argument 79
discrete plot types 455, 449
DISP 258, 332, 344
display 255
freeze 258
graphics 123, 124, 260
mode 525
standard 126, 256
distribute-prefix-operator 567
distribution 546, 547, 566, 568
function 673
divide bar 163
DO loop 236
DO 238
DOERR 100, 104, 241, 242
DOT 282, 288
DOT+ 473
DOT- 474
double guess 430, 431
integral 616
inverse 568
negate and distribute 567
negative 568
quote 36, 62
double-space mode 179
double-where 599
DRAW 53, 249, 271, 417, 449, 452, 453, 459, 461,
464, 469, 478, 480, 488, 489, 493, 495, 497, 498,
501, 502, 507, 509, 511, 619, 654, 692
DRAZ 452, 461, 463
DROP 68, 134
DROP2 69
DROPN 69
depn 454
DUP 70, 82, 134
DUPN 72
Subject Index

D- 566
ECHO 148
EDEQ 413
EDIT 76, 147, 154, 311
editing 554
  menu 147, 154
  program 311
edit/view 148
ELSE 225
else-sequence 225
empty 79
END 225
endless execution 61, 87
ENG 608
ENTER 19, 26, 72, 134, 140
  explicit 140
  implicit 140, 145
αENTER 191
βENTER 191, 265
entry mode 141, 142, 187, 189, 550
  algebraic 141, 143, 207
entry, array 151
entry, text 128
ENTRY 143
environment 12, 126, 132, 408
  plot 12, 126, 465
  standard 12, 408
EQ 118, 410, 413, 414, 420, 450, 453, 478, 509, 642
EQ- 527
equality 223
  logical 223
  physical 223
equation catalog 421, 638
equation 534
EquationWriter 134, 148, 156, 266, 521
ERASE 449, 452, 469
ERR0 241
ERRN 100, 104, 240, 241, 242, 316
Evaluate 240, 241, 242
error 239
  bar 653
  beep 179
  trap 239, 242
  custom 242
error-sequence 240
Evaluate 31, 47, 51, 55, 60, 64, 97, 114, 290, 427, 572, 589, 600
evaluate 16, 25, 31
  algebraic 49
exception 243
  action flag 243
overflow 178, 244, 670, 671
underflow 178, 244, 671, 690
exchange of arguments 69
execution 25, 30, 31, 60, 116
  by address 64
  endless 87
  local name 61
  numerical 52, 55, 56, 58, 598
  postponed 146
  preventing 31
  symbolic 52, 58, 225
exit 234, 225
EXP 570
EXPN 50
expansion 541, 547
EXPS 525, 544, 545, 547, 559, 572, 576, 601
EXPFIT 647
explicit ENTER 140
exponent 33
exponentiated 165
EXPR 170
EXPR= 418, 419, 428, 445
expression 16, 48, 534
  defining 206
  manipulations 541
  rearrangement 521
  simplification 545
EXP 570
extensive manipulations 544
Extremum 429, 438
EXTR 436, 486
F(X) 484
F-test 675
FACT 671
factorial 670, 671
factors 546, 563
fast catalog 179
FC? 223
FC?C 177, 223
FCN menu 436, 483
filled curves 478
FINDALARM 638
first guess 429
FIX 526, 608
flag 55, 175, 220, 333
  -2, 56
  -3, 56, 428, 598
  -15, 285
  -19, 36
  -20, -26, 243-244
  -30, 478, 492
  -31, 499
GO→ 151
grads mode 589
graph screen 126, 255, 260, 266, 465
GRAPH 126, 134, 256, 257, 465, 551
graphics 256
cursor 179, 466
display 123, 124, 260, 465
display of expression 156
object 42, 260
greatest common divisor 345
←GROB 265
GROB 42
guillemets 62
GXOR 263, 267, 270
GXOR 263, 267, 270 mantissa 33
*H 454, 457, 472
HALT Not Allowed 480
HALT 109, 290, 313, 315, 316, 317, 320, 329, 332, 335, 480
helvetica 8, 403
HEX 178
hidden operation 133
parentheses 179
HISTOGRAM 464, 511, 512
HISTPLOT 512
HMS+ 626
HMS 625
←HMS 625
HMS→ 625, 626
home directory 88
HOME 91, 114, 121
HP Solve 53, 138, 196, 201, 409, 523, 524, 530, 532, 692
equation entry menu 412
solver menu 410, 416, 418, 443, 532
variables menu 410
HP 15C 608
HP 17B 13
HP 19B 13
HP 27S 13
HP 28 617
HP 34C 409
HP 35 2
HP 41 2-5, 76, 140, 180, 181, 202, 216, 217, 641
HP 65 2
HP 71B 180
HP 80 409
i 35
identity operations 559, 568
IDN 117, 120, 279
IERR 608, 609
IF structure 135, 225
Subject Index

IF 225
IFERR structure 317, 339
IFT 32, 51, 227
IFTE 32, 51, 227
IM 36
immediate entry mode 141, 143
immediate-execute key 141
implicit integration 615
ENTER 140, 145
GET 117
parentheses 165
implied multiplication 162
Improper Definition 209
IM 568
Incomplete Subexpression 161, 551
Inconsistent Units 684
INCR 117, 119, 237
indefinite loop 230, 234, 235
independent column number 647
RAM 99
variable 415, 417, 459, 492, 646
INDEP 417, 454, 459, 460, 478, 489, 493, 497, 502
index for GET 116
index wrap 179
infinite result 178
action flag 243
infix notation 17
operator 162
inner product 283
input and output 329
input list 294
INPUT 335, 341
insert mode 148
integral 598
derivatives 602
limits 600
numerical 608
programs 615
strategies 611
symbolic 598
with units 694
interactive stack 74, 134, 148
intercept 647
Intercept: 649
intermediate result list 296
intermix binary and real 41
real and complex 36
internal accuracy 33
Invalid Array Element 151
Invalid Card Data 105
Invalid Date error 626
Invalid PTYPE 483
Invalid Syntax 161
Invalid Unit 684
Invalid User Function 210
Inverse of power or inverse of inverse-product 568
INV 568
IOPAR 118
IP 590
IR port 179
ISECT 485, 486
ISOL 207, 249, 439, 519, 521, 523, 525, 526, 527,
530, 531, 541, 598, 689
italics 8, 404
iteration 225, 230
KEEP 76
Kermit message 179
Kermit overwrite 179
key assignment 175
buffer 339
code 341
format 8
menu 8
plane 180
shifted 8
type 141
typing 141, 142, 190
KEY 335, 339, 341
key-per-function 125
keyboard 127
standard 126
keycode 180, 339
keys, format 404
keys, menu 404
keys, shifted 404
KILL 316
known variables 520
label 217
LABEL 462, 473, 478, 493
last arguments 72, 87, 179, 242, 312
command 176
error message 123, 124
error number 123, 124
menu 129
stack 72, 144, 176
LAST 72
LASTARG 72, 87, 242, 460
-LCD 257, 260
LCM 345
least-common-multiple 345
least-squares fit 649, 656
LET 86
LEVEL 76
library 30, 96, 99, 100, 123, 124, 401
Subject Index

attach 101, 102, 104
ID 100
title 102
LIBRARY menu 46, 96, 105, 137
LIBS 104
limits of integration 600
LINE 272, 274
linear 598
coefficient 599
format 156
regression 649
linear congruential method 672
LINE 474
LINFIT 647, 649
LISP 5
list 40, 277, 289
object 25, 30, 32, 51, 289
output 296
-LIST 290, 293
LIST- 291
LNP1 601
LN 570
local memory 109, 110, 123, 124, 245, 248
name 30, 109, 111, 206, 208, 245, 538
name execution 61
name resolution 249
variable 79, 59, 61, 74, 109, 205, 206, 231, 245, 248, 315, 538
variable structure 111, 246
LOGFIT 647
logical coordinates 269, 456
equality 223
operator 221
logical units 460
loop 230
index 231, 246
definite 230, 234
DO 236
endless 61
indefinite 230, 234, 235
WHILE 236-238
loop-sequence 236, 237
LR 345, 509, 649, 651, 653
Jukasiweicz, Jan 17
-M 567
MacLaurin's formula 594
magnitude 679
manual operation 26
mark, plot 468
MATCH 572, 574, 604
tMATCH 525, 572, 593
MATCH 525, 572, 605
mathematical approximation 433
mathematical function 207
matrix 40, 277
MatrixWriter 134, 148, 149, 176, 642
MAXS 645
MEAN 644
MEM 96, 105, 184
Memory Clear 124, 125
memory reset 123, 124, 622
memory, local 109, 110, 123, 124, 245, 248
user 59, 88
VAR 59
MENU 130, 185, 333
menu, custom 185, 175
exit 132
key label 126
port 99
screen 255
subexpression 170
VAR 83, 112, 137, 185, 245
MERGE 105
merging 541, 567
message, prompt 179
message, table 242
minor 307
MINS 645
mode 12, 175, 408
coordinate 285
entry 141, 142, 187, 189
insert 148
numeric 225
program entry 76, 143, 214
server 138
user 123, 124, 129, 158, 180
mode-dependent key 141
Modl: 647
MOD 590
multiple integral 616
plots 454
roots 529
M- 567
name, global 30, 59, 60, 61, 83, 90, 245, 327
local 30, 109, 111, 206, 208, 245
object 29, 58, 116, 233
port 97
quoted 62
resolution 90, 110, 249
XLIB 30, 46, 61, 103, 181
NEG 260, 262
negative pixel coordinates 274
NEG 568

- 711 -
Subject Index

NEW 413
newline 151
NEWOB 100, 297, 300
next event 622
non-analytic function 26
Non-Empty Directory 94, 96
Nonexistent Alarm 637
normal distribution 673
normal-sequence 239
NOT 238
notation 8, 403
common 17
infix 17
Polish 17
prefix 17
Reverse Polish 15
~NUM 32, 55, 56, 427, 598, 608, 617
NUM 39
numbered register 59
numerator 163
numerical execution 52, 55, 56, 58, 598
integration 608
mode 225
numeric/symbolic execution 178
NXEQ 485
object 25, 139
class 32
composite 30, 51, 63, 289, 299
data 32
definition 27
entry 127
graphics 42, 260
name 29, 58, 116, 233
program 30
string 36, 144
symbolic 63
system 27
tagged 42, 345
type 27
unit 44
untagged 43
value 27
Object In Use 99, 104
object-to-grob conversion 265
OBU~ 35, 38, 43, 44, 69, 112, 191, 278, 288, 291,
574, 578, 605
obtaining guesses 433
OCT 178
operation 25, 59
manual 26
logical 221
ORDER 85, 112, 422, 423
output, display 344
list 296
OVER 71
overflow 178, 244, 670, 671
Owner's Manual 5, 402
\( \pi \) 56, 211
page, menu 129
PARAMETRIC 464, 497
parent 90
parsing 144
past due alarms 628
path, current 89, 101
path name 113
PATH 89, 113
PDIM 267, 458
pencil-and-paper 18
pending alarm 628
permanent custom menu 185
PERM 669
PGDIR 96, 106
physical equality 223
PICK 70
PICT 267, 476
picture 255
PIX? 272
PIXOFF 271, 274
PIXON 270, 274, 473
plot cursor 466
environment 12, 126, 408, 451, 465
resolution 460
ranges 692
with units 692
PMAX 456
PMIN 456
polar coordinates 34, 284
plots 491
POLAR 464, 491
Polish notation 17
polynomial fits 662
polynomialize 576
polynomials 575
port 96, 97, 99, 100, 105
menu 99
name 97
variable 97, 96
POS 39, 293
postponed execution 146
PPAR 118, 138, 269, 270, 454, 456, 458, 459, 460,
461, 464, 507, 511, 647
PR1 138
precedence 17, 179, 565
predicate operator 689
Subject Index

PREDX 650
PREDY 650
prefix notation 17
operations 567
operators 567
unit 681
preventing execution 31
PRG annunciator 141
principal value 178, 531
printer port 179
probability density 673
probability menu 669
problem solving 195, 198, 516, 520
procedure 47, 200, 201
defining 11, 246
program 29, 47, 52, 196, 201, 213
as argument 331
body 214, 215
content 47
definition 47
editing 311
entry mode 76, 141, 143, 201, 214
legibility 205
object 30
optimization 323
quotes 62
quoted 63
structure 47, 202, 213, 215, 219, 326
structure word 32, 142, 213, 219
suspended 313, 329
unquoted 63
programming 195, 202
structured 203, 216, 217, 313
prompt message 179
PROMPT 290, 313, 315, 317, 320, 329, 330, 332, 335, 344, 480
PRTPAR 118
PRVAR 97, 318
pseudo-linear fits 650
pseudo-linear regression 650
PTYPE 448, 489
PURGE 86, 92, 96, 97, 99, 100, 267
directory 94
recovery 87
PUT 117, 119, 120, 186, 278, 281, 291, 295
PUTI 117, 119, 120, 278, 291
PVARS 96
PVIEW 257, 270, 345
PWRFIT 647
PX- 270, 274
→Q 193
QUAD 249, 489, 519, 521, 525, 526, 528, 530, 531, 541, 693
quadratic equations 528
qualifying message 412, 427, 428
quotation mark delimiters 62
quote, name 62
program 62
single 62
string 62
tagged object 99
quoted argument 535, 617
QUOTE 535, 538, 601
→Q 525, 608
→Qπ 525, 526
radians mode 570, 589
random number 671
RAND 648, 649, 650, 671
RATIO 164
RCEQ 138, 452
RCL 60, 97, 100, 113, 116, 222, 267
RCLALARM 183, 628, 637, 640
RCLF 177
RCLKEYS 183
RCLMENU 186
RCWS 178
RDM 117, 120, 278
RDZ 672
RE 36
real number 33
recentering 469
Recover RAM 124, 125
recovery 314
argument 72, 87, 179, 242, 312
from PURGE or STO 87
stack 72
rectangular coordinates 284
referenced 99
register 115
numbered 59
storage 59
reordering terms 541
repeat interval 628
REPEAT 237, 238
REPL 39, 172, 264, 267, 268, 270, 292
REPL 476, 545
Replace RAM, Press ON 105
reschedule 179
RESET 451, 455
resolution, name 110, 460, 477, 507
local name 249
RESTORE 106, 107
result 16
RES 454, 499
Subject Index

Reverse Polish Notation 15
RE 436, 568
right hand 9, 405
RND 690
ROLL 69
ROLLD 69
root 409, 693
ROOT 201, 249, 413, 426, 438, 440, 482, 485
root-finder 409
ROT 70
row number 116
order 277
vector 282
RPL 5, 12, 21, 22, 60, 64, 82, 112, 518
RPN 3, 15
command 26
principle 15, 50
RULES 132, 149, 157, 172, 522, 525, 544, 545, 550, 556, 557, 560, 561, 572, 575
R→C 35
Σ 183
Σ 587, 595, 601
Σ + 510, 660, 641
Σ − 641
SAME 44, 223
sample covariance 648
standard deviations 645
saving, argument 87
SCALE 454, 457
scatter plot 507, 653
SCATTER 464, 507, 654
SCI 608
SCL 508
Sconj 117, 120
screen 126, 255
graph 126, 255, 260, 266, 465
menu 255
text 126, 255, 260
ΣDAT 118, 453, 507, 510, 642
SDEV 645
selection arrow 421
environment 550
separator 144
sequence 9, 213, 230, 233, 405
server mode 138
set flag 175
setting an alarm 628
SF 55, 176, 222
SHOW 425, 439, 440, 528, 537, 599, 694
signal flag 176, 244
SIGN 499, 690
simple units 682
simplification, automatic 52, 57
SIN 175
single guess 430, 431
single quote 62
single-step 316, 319
SINV 117, 120
SIN 589
SI 680
SIZE 38, 262, 267, 278, 291
SKEY 184
SKIP 147
→SKIP 147
slope 647, 649
Slope: 649
SNeg 117, 120
solve variables menu 197
solver menu 197, 410, 418, 418, 443, 532
modifying 443
space, in EquationWriter 161
ΣPAR 118, 509
spectral test 672
spherical polar coordinates 284, 286
square root 165
SQ 508
SST 316, 319
stack 9, 19, 21, 67, 125, 405
diagram 10, 406
flag 177
level 29, 67
recovery 72, 312
roll 69
interactive 74, 134, 148
unlimited 77
display 126, 256
environment 12
keyboard 126
standard environment 12, 408
start 230, 231, 233
START...NEXT 234
START...STEP 234
starting and stopping 313
statistics 641
catalog 642
matrix 149, 641
one-variable 644
summary 655
two-variable 646
status area 126
STD 525
step 233
step-wise substitution 60
STEQ 138, 410, 413, 452
STO 44, 83, 87, 92, 97, 100, 267, 268, 281, 476, 642
STO recovery 87
STO* 118
STO – 118
STO+ 118
STO/ 118
STOALARM 630, 636, 640
STOF 177, 183
STOKEYS 182
STOΦ 642, 644
stop 230, 231, 233
storage arithmetic 118
storage register 59
→STR 38
string object 36, 144
counted 37
stripping tags 44
structure 50
    local variable 111, 246
    program 47, 202, 213, 215, 219, 326
    word, program 32, 142, 213, 219
structured programming 203, 216, 217, 313
STR– 112
STWS 41, 178
SUB 39, 265, 266, 267, 292
subdirectory 90, 420
subexpression 50, 158, 170
    cursor 170
    environment 521
    level 50
    menu 170, 551
    mode 550
    closing 160
subroutine 217, 330
substitution 64, 523
SUB 476, 545, 603
summands 546
summary statistics 655
summation 231, 595
patterns 596
suspended program 313, 329
SWAP 69, 70, 134
symbolic array 301
    calculator 3
    constant 57, 170, 178
    execution 52, 58, 225, 598, 598, 608
    fraction entry 163
    integration
    manipulations 196
    math 197, 515
    object 63, 515
    solutions 439, 524
syntax 140
    algebraic 16, 48
SYSEVAL 64
system flag 175
system halt 88, 100, 123, 124, 622
System Internationale 680
system object 27
→T 560, 563
T→ 563, 560
%T 591, 689
table-filling evaluation 427
tag 42
    stripping 44
    →TAG 43
tagged object 42, 345
    object, quoting 99
Taylor's polynomial 594, 617
TAYLR 529, 579, 587, 694
temporary custom menu 186
term movement operations 560
test 221
    command 221, 223
test-sequence 225, 236, 237
text entry 128
    screen 126, 255, 260
TEXT 126, 257
THEN 225
then-sequence 225
3D 287
tick marks 463
ticking clock 179, 621
ticks 62, 622, 624, 636
TICKS 623
time arithmetic 625
    menu 621
    numbers 625
    number 622
    system 621
    value of money 409
TIME 623, 624
~TIME 623, 624
time-adjust menu 623
time-set menu 623
TLINE 272, 274, 474
TMENU 130, 186, 333
Too Few Arguments 210
transformation 555
trigonometric functions 570, 691
triple guess 430, 431
TRN 117, 120, 279
TRNC 690
truth plot 501
Subject Index

truth-valued function 501
TRUTH 464, 501
TSTR 624
TVARS 121, 122
TVM 409
24-hour format 179, 622
2D 35, 287
two-variable statistics 646
type number 27
TYPE 27, 44, 52, 63
type, object 27
type-ahead 339
typing 188
key 141, 142, 190
UBASE 224, 442, 682
UFACT 224, 690
Unable to Isolate 527
unconditional branch 225
Undefined Local Name 538
Undefined Name 53, 573
underflow 178, 244, 671, 690
uniformly distributed 671
unit built-in 682
conversions 684
in custom menu 687
differentiation 693
integration 694
magnitude 44
management 44, 679
menu 685
object 44, 679
plotting 692
prefixes 681
simplifications 689
-UNIT 684
unknown variable 415, 417, 520
unlimited stack 77
unquoted program 63
untagged object 43
UNTIL 236
UPDIR 90
upper-tail distribution function 673
user annunciator 180
flag 176, 222
key action 179
key assignment 180
memory 59, 88
mode 123, 124, 129, 158, 180
1USR 180
user-defined derivatives 590
user-defined function 86, 161, 196, 204, 205, 210, 288, 538, 590, 592, 600
user-defined units 683
UTPC 674, 675
UTPF 674, 676
UTPN 674
UTPT 674, 675
UVAL 684
-V2 35, 287, 288
-V3 287, 288
value, object 27
VAR 645
VAR menu 83, 112, 137, 185, 245, 410, 421, 532
VAR memory 59
variable 16
formal 61, 249
global 45, 59, 83, 111, 245, 248, 329
local 79, 59, 61, 74, 109, 205, 206, 231, 245, 248, 315
port 97, 96
VARS 121, 122
VEC 154
vector 40, 277, 282
contravariant 282
covariant 282
vectored ENTER 144, 145, 179, 190
vertical lines 478
VIEW 76
VISIT 76, 147, 311
VTYPE 27, 52, 122
V→ 35, 287
*W 454, 457, 472
WAIT 341
where 168, 599, 617
WHILE 238
WHILE loop 236-238
-WID 150
WID→ 150
wild card names 572
wordsize 41
χ² distribution 675
xₘₐₓ 454
xₘᵢₙ 454
xₐₓᵉˢ 454
XCOL 453, 507, 509, 511, 512, 646
XLIB name 30, 46, 61, 103, 181
XRNG 454, 457
XROOT 160, 166, 690
yₘᵢₙ 454
yₘₓ 454
yₐₓᵉˢ 454
YCOL 453, 507, 646, 649
YRNG 454, 457
Zero 419, 428
Delimiters and Punctuation

" " 36, 62
# 41
[ ] 40
<> 213, 214
(): 43
, 44
' ' 62
{ } 40
\ 24, 285
@ 36, 146
\rightarrow 206, 210, 246, 290
? 687
\rightarrow 9, 405

Functions

+ 262
! 671
% 590
\int 596, 598, 608
\partial 538, 587, 590, 694
\sqrt 165
| 160, 168, 599
= 224
== 223, 224
\neq 223
> 223
< 223
\leq 223
\geq 223
\neq 223

RULES Operations

\neg A 565
A\neg 565
AF 569
COLCT 572
\neg D 566
D\neg 566
DINV 568
DNEG 568
E^* 570
E() 570
L\# 570
L() 570
\neg M 567
Subject Index

M→ 567
T→ 563
T- 563
TRG* 570
→TRG 570
(← 566
→) 566
( ← 565
(0) 566
→( ) 567
( ) 567
1/( ) 568
+1−1 569
/1 569
*1 569
^1 569
The HP 48S/SX Scientific Calculators provide powerful symbolic and numerical computing features that are applicable to a wide range of problems. There is special emphasis on six application areas, with special prompting and operating environments to facilitate interactive manual calculations. *HP 48 Insights II* focuses on these areas, which are indicated by the legends above the top six keys of the number pad: SOLVE, PLOT, ALGEBRA, TIME, STAT, and UNITS.

This book is the second volume of a two-part series by Dr. William Wickes on the operation and application of the HP 48. *Part I* concentrates on the underlying unified principles of HP 48 operation, and the tools and techniques for programming the calculator, including calculation methods, object storage, display management, and customization. *Part II* focuses on problem-solving in the six special application areas, including discussions of design motivations, “how-to” descriptions and examples for interactive operation, and methods for extending the built-in functionality for custom applications. More than 60 programs are included as examples and extensions, including two multi-program systems for polynomial manipulations and least-squares fitting.

*Part II* of *HP 48 Insights* is self-contained, and can be appreciated by anyone with a rudimentary knowledge of the HP 48. *Part I* is not a prerequisite, although many of its principles and methods are applied in *Part II*. Together, the two volumes constitute the most comprehensive and effective exposition available of the principles and methods of the remarkable HP 48.

**Chapter Headings for Part II:**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>Introduction to Part II</td>
<td>401</td>
</tr>
<tr>
<td>14.</td>
<td>HP Solve</td>
<td>409</td>
</tr>
<tr>
<td>15.</td>
<td>Plotting</td>
<td>447</td>
</tr>
<tr>
<td>16.</td>
<td>Symbolic Objects and Solutions</td>
<td>515</td>
</tr>
<tr>
<td>17.</td>
<td>Expression Manipulations</td>
<td>541</td>
</tr>
<tr>
<td>18.</td>
<td>Calculus</td>
<td>587</td>
</tr>
<tr>
<td>19.</td>
<td>The Time System</td>
<td>621</td>
</tr>
<tr>
<td>20.</td>
<td>Statistics</td>
<td>641</td>
</tr>
<tr>
<td>21.</td>
<td>Unit Management</td>
<td>679</td>
</tr>
</tbody>
</table>

ISBN 0-9625258-4-7