

# **CALCULUS WITH THE HP48**

**SECOND EDITION**



**LYNN E. GARNER**







# **Calculus with the HP 48**

**Second Edition**

**Lynn E. Garner**  
**Brigham Young University**

**Dellen, an imprint of**  
**Macmillan Publishing Company**  
*New York*

**Maxwell Macmillan Canada**  
*Toronto*

**Maxwell Macmillan International**  
*New York Oxford Singapore Sydney*



© Copyright 1993 by Macmillan Publishing Company,  
a division of Macmillan, Inc.,  
Dellen is an imprint of Macmillan Publishing Company.

Printed in the United States of America.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the Publisher.

Macmillan Publishing Company  
866 Third Avenue, New York, New York 10022

Macmillan Publishing Company is  
part of the Maxwell Communication  
Group of Companies.

Maxwell Macmillan Canada, Inc.  
1200 Eglinton Avenue East, Suite 200  
Don Mills, Ontario M3C 3N1

ISBN: 0-02-340582-1

Printing: 1 2 3 4 5      Year: 3 4 5 6 7



## Preface

There are calculators, scientific calculators, graphing calculators, and the HP 48S and HP 48SX, the symbol-manipulating graphing calculators. The HP 48 represents the state of the art in calculators, even blurring the line between calculators and computers.

The HP 48 is a stack-based machine, meaning that operations are performed on objects already on the stack. That forces "reverse Polish" logic, notable for the fact that no parentheses are needed in performing complex arithmetic. The 48 handles 20 different data types, including numbers, complex numbers, vectors, matrices, strings, lists, algebraic expressions, graphics objects, and programs, as well as managing units (148 plus compounds) and storing objects in a multi-level menu.

The HP 48 has 32K of RAM and 512K of ROM. It has 2100 built-in commands organized into 60 menus, with 58 commands on the keyboard. It has a 224-character symbol set, 150 accessible from the keyboard. The entire keyboard also has a "user" mode, in which another 270 key assignments can be made. For exchanging files, it has an infrared read/write port and an RS232 port, with Kermit in ROM; serial interface kits for desktop computers and printers are available.

Built in are:

- numerical routines for root finding, numerical integration, and finite summations;
- algebraic manipulations for expanding, collecting, equation-solving, derivatives, Taylor polynomials, and some antiderivatives;
- operations for complex numbers, vectors, and matrices, including inverses, determinants, and various norms;
- statistical routines for standard one- and two-variable statistics, including covariance, correlation, regression (5 models), and upper tail probabilities;
- graphic capabilities for 8 types of functions, graphical analysis (zeros, slopes, extrema, derivative, and area), multiple graphs, some painting, and animation;
- a programming language (RPL) with such high-level constructs as counted and indefinite loops, branching structures including case statements, and a debugging environment;
- editing environments, including special ones for equations and matrices;
- a built-in clock with alarms, date and time arithmetic, and timed start-up; and
- a programmable BEEP command that allows one to play music.

It is not the purpose of this monograph to explain all the features of the HP-48. Rather, its aim is to give the student a brief tutorial in its basic mathematical uses, and then to explore the many ways in which the HP 48 can be used to enhance a calculus course. In the process, the student will become familiar with most of its mathematical features.

Neither is it the purpose of this monograph to teach calculus. We assume that the student is looking for a way to speed up and enhance the learning of calculus concepts. The traditional topics of the three-semester "engineering and science" calculus course are dealt with here, as well as special environments suitable for the various approaches of "reformed" calculus. Ways to use the calculators in connection with each of the topics are presented.



Therefore, this book is a collection of tutorial helps, tips, tricks, explanations, programs, and environments that will aid the calculus student to "get the most" from his investment in the state-of-the-art technology represented by the HP-48S and HP-48SX.

It has been my pleasure to associate with many students who have come to love the HP 48 as I have. To them for the insights they have given me and for letting me experiment with their classes I express my gratitude. I am also grateful for the support and encouragement rendered by the people at Hewlett-Packard in Corvallis. And to my colleagues from across the country who have helped me understand better how a calculator can be used appropriately in a math class I am particularly indebted.

Lynn E. Garner  
Brigham Young University



# Table of Contents

<b>Chapter 1. Basic Use Tutorial</b>	1
The Keyboard	1
The Stack	1
Entry	2
Operations	2
Stack Manipulation	3
Data Types	3
Menus	4
Soft Keys	4
Modes	4
The VAR Menu	4
Storing	4
Recalling	5
Purging	5
Renaming	5
Editing	5
Ordering	6
Subdirectories	6
Navigation	6
Form of a Number	7
Rounding: Number of Decimal Places	7
Rounding: Number of Significant Digits	7
Decimals to Fractions	8
Prime Factorization	10
<b>Chapter 2. Use in Precalculus Topics</b>	13
Algebraic Manipulation	13
Solving Equations	13
Isolating a Variable	13
Root Finding	14
Evaluating Expressions	14
Graphs of Functions	16
Plot Parameters	16
Drawing	17
Saving	18
Zooming	19
Multiple Graphs	19
Zeros and Other Graphical Properties	19
Plotting Unusual Functions	19
ABS and FLOOR	19
Fractional Exponents	20
Routines as Functions	20
Animation	21
Synthetic Division	21
Trigonometric Functions	22
Angle Modes	22
$\pi$	22
Values	23
Inverse Trig Functions	23
Applications of Trigonometry	25

Logarithmic and Exponential Functions.....	28
Linear Algebra .....	28
Vectors.....	29
Matrices and Determinants .....	30
Systems of Linear Equations .....	31
Gauss-Jordan Reduction.....	31
Combinatorics.....	34
Elementary Plane Analytic Geometry.....	35
Distance Formula.....	35
Midpoint Formula.....	35
The Line on Two Points.....	35
The Point on Two Lines.....	36
Collinear Points .....	37
Concurrent Lines .....	37
Distance from a Point to a Line .....	38
<b>Chapter 3. Limits and Derivatives .....</b>	<b>39</b>
Limits.....	39
Graphical Evaluation .....	39
Computational Evaluation .....	39
Finding Derivatives.....	39
Zooming .....	39
Simplifying.....	40
Evaluating .....	40
Higher-order Derivatives .....	41
Implicit Differentiation.....	41
Tangent Lines .....	42
Function Analysis.....	43
Points of Inflection .....	43
Root Finding.....	43
Bisection Method.....	43
Newton's Method.....	45
Function Tables .....	46
Creating Tables of Function Values.....	47
Difference Tables.....	48
<b>Chapter 4. Integrals and Their Applications .....</b>	<b>51</b>
The Built-in Integrator .....	51
Areas, Volumes, and Arc Lengths .....	52
Antiderivatives.....	53
Differential Equations .....	54
Slope Fields.....	55
Integral Curves.....	56
Boundary Value Problems.....	57
<b>Chapter 5. Transcendental Functions .....</b>	<b>59</b>
LN, e, and EXP .....	59
Inverse Trigonometric Functions .....	59
Hyperbolic Functions .....	59
Applications.....	60
Curve Fitting.....	60
Functions with Parameters.....	61
Plotting Data Points.....	62
Fitting Curves to Data .....	62



The Line on Two Points.....	62
Least Squares Curves.....	63
Cubic Splines.....	64
<b>Chapter 6. Numerical Integration Theory .....</b>	<b>67</b>
Riemann Sum Rules .....	68
Error Analysis.....	69
Trapezoidal Rule.....	70
Midpoint Rule.....	71
Simpson's Rule.....	71
<b>Chapter 7. Sequences and Series.....</b>	<b>73</b>
Terms of a Sequence.....	73
Sequences by Formula.....	73
Recursive Sequences.....	74
Partial Sums.....	75
Geometric Series.....	75
Taylor Polynomials.....	75
<b>Chapter 8. Conic Sections and Polar Coordinates .....</b>	<b>79</b>
Graphs of Conics.....	79
Implicit Functions.....	79
Rotation of the Coordinate System .....	80
Parametric Equations .....	82
Chain Rule.....	82
Graphs .....	83
Polar Coordinates .....	83
Graphs.....	83
<b>Chapter 9. Solid Analytic Geometry .....</b>	<b>85</b>
Points and Planes.....	85
The Plane on Three Points.....	85
Distance from a Point to a Plane .....	85
Lines.....	86
Line on Two Points.....	86
Line of Intersection of Two Planes .....	87
Space Curves and Graphs.....	88
Parametric Surfaces .....	91
Planes .....	91
Graphs of Functions of Two Variables.....	92
Quadric Surfaces .....	92
Surfaces of Revolution.....	93
Graphing Parametric Surfaces .....	94
Vector Functions.....	96
Velocity and Acceleration.....	97
Curvature .....	98
Cylindrical Coordinates .....	98
<b>Chapter 10. Partial Differentiation .....</b>	<b>101</b>
Functions of Two Variables.....	101
Graphs.....	101
Contour Maps.....	101
Partial Derivatives .....	101
Two-Variable Newton's Method .....	101

Gradients .....	103
Directional Derivatives .....	104
<b>Chapter 11. Path Integrals and Multiple Integrals.....</b>	<b>107</b>
Path Integrals .....	107
Numerical Multiple Integration .....	108
Simple Integrals .....	109
Double Integrals .....	110
Triple Integrals.....	112
<b>Chapter 12. Just for Fun .....</b>	<b>115</b>
The Time Value of Money .....	115
Chaos.....	116
Public Key Cryptography .....	119
Music.....	123
<b>Index.....</b>	<b>125</b>



# **Calculus with the HP 48**









## **Chapter 1. Basic Use Tutorial**


The HP 48 (S or SX) represents the world standard in a calculator. It is powerful enough to provide sophisticated learning environments, and at the same time simple enough to get quick and accurate answers to simple problems. It is an ideal machine for studying calculus—it is always at hand and it can do anything you want to do in calculus. It is a "learning machine" that the student can gradually transform into a "doing machine" suitable for the professional in the field.


Our main point of view here is to learn the principles that govern use of the calculator. The *HP 48SX Owner's Manual* (hereafter called the *Manual*) provides many examples, but it sometimes fails to tell you the principles involved. Hopefully, by learning the overriding principles, you will quickly master your machine.

### **The Keyboard**

Take some time to study the keyboard of your HP 48. Note the locations of the various types of keys: the number keys, the letter keys, and the operation keys. Note other symbols that occur as labels; you will need them eventually. Many of the keys are menu keys, but you can't tell by looking in all cases; you will learn them as we go along.

Of particular importance are the shift keys. On the HP 48 there are three keys that behave as shift keys. The blue key is called the *right-shift key*, and is used to get the blue labels; the orange key is called the *left-shift key*, and is used to get the orange labels. The  key is used to get the letters. While the  key is held down with one finger, letters may be typed with another. Pressing  twice "locks" it so that several letters can be typed in succession without holding down the  key. Pressing  again or pressing  unlocks the  $\alpha$ -shift.

The right- and left-shift keys are used with the  key to get other symbols, including lower-case letters and Greek letters, that are not printed on the face of the machine; see pp. 52ff of the *Manual*. The HP 48 has 49 keys with 230 symbols or commands assigned to them. Of the 224 characters in the HP 48 character set, 150 are directly accessible from the keyboard.

The HP 48 also has a User keyboard, activated by the  command, that is completely blank; you can assign to the keys anything you want, as described starting on p. 216 of the *Manual*. A total of 276 key assignments can be made on the User keyboard.

### **The Stack**

The HP 48 is a *stack-based* machine, meaning that the basic method of handling data is by use of a *stack*. Each new item that is entered into the machine becomes the bottom object on the stack, and all other objects already on the stack are pushed up to the next higher positions.



Commands given to the machine are applied to objects already on the stack. For example, the command +, to add, is interpreted by the machine to mean that the bottom two objects on the stack are to be taken from the stack and added, and then their sum is to become the new bottom object. Since two objects are taken off the stack, and only one is placed back on, all other items on the stack will drop down one position.

## Entry

To enter an item into the machine, simply type it in. As you type, the object appears below the stack in what is called the *command line*. The end of typing is signaled by pressing **ENTER**; that command takes the object from the command line and places it on the bottom (level 1) of the stack.

If you make a mistake while typing an item, the back arrow key **←** will erase one character at a time. The cursor keys may also be used to reposition the cursor on the command line; the **DEL** key erases the character on top of which the cursor rests. To erase the entire object you are typing, before you press the **ENTER** key, press the **ATTN** key (the **ON** key).

To remove the bottom object from the stack, use **DROP** (the **←** key performs that function without shifting if there is no command line). **DROP** may be used repeatedly to remove several objects from the stack, one after another. To remove all objects from the stack, press **CLR**.

## Operations

As was mentioned above, each command given to the machine applies to the objects already on the stack. The buttons **+**, **-**, **×**, and **÷** command the machine to take the bottom two objects from the stack, add, subtract, multiply, or divide, respectively, and put the result back on the stack. Thus to perform an operation on two numbers, you must first place the two numbers on the stack. Actually, most commands will also enter the second number for you, making it unnecessary to press **ENTER** after the second entry.

For example, to multiply 21 by 56, type

21	<b>ENTER</b>	56	
		<b>×</b>	

1:	21	
56		
<b>GRAPH NUMN GEOM CALC SERIE PPMY</b>		
2:		
1:	1176	
<b>GRAPH NUMN GEOM CALC SERIE PPMY</b>		

This order of doing things is what is called "reverse Polish" logic.

Other commands work the same way, though they may not operate on exactly two stack objects. The exponentiation command  $y^x$  and the root command  $\sqrt[x]{y}$  take two numbers from the stack, but the  $1/x$ ,  $\sqrt{x}$ , and  $x^2$  commands operate on only the bottom number, the one in level 1. Still other commands that we will meet take many objects from the stack, and some do not use stack objects. For example,  $\text{CLR}$  uses as many stack objects as there are, and  $\text{OFF}$  ignores the stack.

### Stack Manipulation

$\text{DROP}$  and  $\text{CLR}$  are examples of stack manipulation commands; that is, they only move (or remove) the objects on the stack. Another stack manipulation command on the keyboard is  $\text{SWAP}$ , which, as you might expect, interchanges the bottom two objects on the stack.

Another useful command is to press  $\text{ENTER}$  to duplicate the object on the bottom of the stack. It is equivalent to the  $\text{DUP}$  command found on the PRG STK menu; DUP is used in a program to duplicate the bottom object.

Other stack manipulation commands are found in the PRG STK menu, described in the *Manual* on p. 78.

### Data Types

The operations and manipulation commands given above apply not only to numbers on the stack, but other objects as well. The HP 48 has 20 data types, the most useful of which are numbers, arrays (matrices), vectors, complex numbers, algebraic objects, lists, strings, graphics objects, and programs. All data types are available for use at all times, even from programs, which makes the calculator very powerful. Operations on objects requires compatibility of data types, however; not just any two things can be added, for example.

To give you an idea of how different objects are treated by the same command, we will consider the  $+$  command, addition. If real numbers, complex numbers, or binary integers occupy both the bottom levels of the stack,  $+$  has the anticipated effect: it replaces them with their sum. The same is true of vectors and matrices, provided that the arrays have the same sizes; an array sum is a new array whose entries are the sums of the entries in the old arrays. For strings and lists,  $+$  means concatenate; for example, if "AB" is added to "CDE", the result is "ABCDE". For names and algebraic expressions,  $+$  yields the indicated sum, which is presented as an algebraic expression. If  $+$  is applied to two graphic objects, the objects are superimposed.  $+$  will not operate on programs.

Data types are described starting on p. 80 of the *Manual*. More information on operations is given starting on p. 133.

## Menus

On the HP 48 there are a total of 60 menus, counting submenus, listed on pp. 697-8 of the *Manual*, and discussed briefly on pp. 55ff.

Most menus have more than six commands, the maximum number that can be visible at one time. To get the next page of commands, press the **NXT** key; **PREV** gives the previous page of a menu.

## Soft Keys

The top row of keys under the calculator's display window are called *soft keys* and are used to select commands from a menu. Whatever menu is showing, each soft key has the meaning of the command in the panel of the menu above that key. When a menu changes, the meaning of the soft key also changes.

## Modes

The MODES menu allows you to select the notation in which numbers are presented, whether you want angles in radians or degrees, and so forth. Its commands are discussed in the *Manual*, starting on p. 220. It is probably good to start by selecting STD, SYM, STK, ARG, CMD, ML, and RAD. If at any time the machine fails to perform as you expect, check to see that the modes have not been reset. (Some games change mode settings; many students have been puzzled at the machine's refusal to draw graphs after they have been playing *Tetris*.) Most of the things you want to do in calculus require that SYM (symbolic) and RAD (radians) modes be set.

## The VAR Menu

The VAR menu is the storage place for containers that you create. Whenever an object is stored in a container other than the stack, it appears on the VAR menu. Press **VAR** to view the containers in the VAR menu.

## Storing

To store an object, put the object on the bottom of the stack. Select a name for the object, and type in the name, beginning with the single quote mark, '. Then press the **STO** button, and the object is stored in a container with the name you typed.

For example, to store the object on the bottom of the stack in a container named AB, just type

' A B

(type ' first, then hold down **α** and type  
A and B, then release **α** )

and press **STO**.

1: 1176  
'AB'  
GRAPH NUMN GEOM CALC SERIE PPRCT

1:  
AB GRAPH NUMN GEOM CALC SERIE



The name AB will appear in the VAR menu.

### Recalling

There are several ways to recall the contents of a container. If the container does not contain a program, simply pressing the soft key under the name of the container will put the contents of the container on the stack. If the container does contain a program, however, pressing the soft key will cause the execution of the program. Therefore, a program must be recalled using `[RCL]`. The easiest way is to press the quote mark, `'`, and then press the soft key corresponding to the container, thus putting the name of the container on the stack. Then press `[RCL]`. The contents of the container is placed on the stack. Using `[RCL]` also works with containers of objects other than programs. A shortcut for `[RCL]` is to press the right-shift key and then the soft key corresponding to the container.

### Purging

To remove a container (and its contents) from the VAR menu, type the name of the container, beginning with `'`, and then press `[PURGE]`. To remove several containers at once, make a list of their names, separated by spaces, and then use `[PURGE]`. Press `[{ ]` to start a list and then press the soft keys of the containers that you want to remove. When the list is complete, press `[ENTER]` and then `[PURGE]`.

### Renaming

To change the name of a container, recall the contents and store it under the desired name, and then purge the old container. There is no provision for renaming in any other way. This procedure works equally well with variables and with subdirectories.

### Editing

There are several ways to change the contents of a container. The first, which is a complete change-over, is simply to store a new object in the old container.

A second way to change the contents of a container is to "visit" the container and edit its contents. First type the name of the container, starting with `'` (or type `'` and then press the soft key of the container), and then press `[VISIT]`. The contents of the container is placed in editing mode, and the cursor can be moved by pressing the arrow keys; the `[DEL]` key deletes the symbol the cursor covers. If you type with the block cursor, previous text is written over; pressing `[INS]` changes the cursor shape to an arrow, after which new text is inserted between previous text. Additional editing mode features are described in the *Manual*, starting on p. 111.

At the end of editing, pressing `[ON]` discards the edited object; the original contents of the container are unaltered. Pressing `[ENTER]` saves the edited object in the container, replacing the old one.

To edit the object in level 1 on the stack, press **EDIT**. To edit the object in level  $n$  on the stack, type  $n$  and then press **VISIT**. Pressing **ENTER** saves the edited object, and pressing **ON** discards it, leaving the original unchanged.

Additional editing environments for equations and matrices on the HP-48 are described in the *Manual* on pp. 241 and 350.

### Ordering

Whenever a new container is created, it appears in the first position on the VAR menu. Therefore, after several containers have been created, you usually discover that you want them in another order. Fortunately, they can be reordered, using the **ORDER** command on the MEMORY menu.

To use the **ORDER** command, first create a list of the menu items in the order in which you want them to appear on the menu. This applies to the entire directory, not just the first six items. Make the list by pressing **{ }** and then the soft keys in the desired order; end by pressing **ENTER**. Once the list is created, call up the MEMORY menu and press **ORDER**; when you return to the VAR menu, the items you specified will be in the order you specified. If the list you make does not specify all the items in the menu, the unspecified items retain their relative order, and are grouped together after the specified items.

### Subdirectories

It is soon apparent to the casual observer that the VAR menu can become quite unwieldy. For that reason, it is convenient to create subdirectories, into which related containers can be organized.

To create a subdirectory, type the name you have chosen for it, and then press **CRDIR** (for "create directory") on the MEMORY menu. When you return to the VAR menu, you will find the name you typed (or as many letters of it as will show) as first item on the menu. It will also look like a file folder, indicating that it is a subdirectory. Enter the subdirectory by pressing the soft key below it. You will see a blank menu, waiting for containers to be created. Anything you store at this point will be stored in the subdirectory. You are now down one level from the top in the VAR menu. The top level is called the HOME level; you can return to it by pressing the command **HOME**.

You can also create subdirectories inside subdirectories. Storing takes place in the current level of the VAR menu. You can use the contents of any container in the current subdirectory, or from any container in any subdirectory "up the line", all the way to the HOME level, simply by calling its name. Programs stored in the current subdirectory or in any "parent" subdirectory, all the way up to the HOME level, can be called at any time.

### Navigation

With several subdirectories, some of them nested within others, navigation around the VAR menu can become a problem. The current path, listing the subdirectories from the HOME

level down to the current level, is shown at the top of the display whenever the stack is visible. The PATH command puts the path on the stack.

If you enter any name from the current path, *without* the single quote mark this time, it is interpreted as a command to go to that subdirectory.

The command `[UP]` moves you up one level in the current path, to the parent of the current directory. In a program, this command is known as UPDIR.

## Form of a Number

Here we discuss some utilities that change the form of a number in the machine. Some are built in, and others we must supply for ourselves.

First of all, study the description of the commands STD, FIX, and SCI on the MODES menu, beginning on p. 220 of the *Manual*. These commands determine how your calculator will customarily present numbers to view.

### Rounding: Number of Decimal Places

The command RND on the MTH PARTS menu rounds the number on level 2 of the stack to the number of decimal places specified by the number on level 1. The "number" on level 2 can be real, complex, a vector, or a matrix. As an example of the use of RND, put your calculator in STD mode, and suppose that you want to invert the matrix  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ .

Enter the matrix by typing `[MATRIX] 1 [SPC]`  
`2 [ENTER] [▼] 3 [SPC] 4 [ENTER] [ENTER]`.

```
1: [[ [ 1 2 ]
      [ 3 4 ] ]]
GRAPH NUM.MN GEOM CHLC SEXIE PPRMT
```

Invert it by pressing `[1/x]`. The result involves numbers with 11 decimal places showing.

```
1: [[ [-1.99999999998 ...
      [ 1.49999999999 -... ] ]]
GRAPH NUM.MN GEOM CHLC SEXIE PPRMT
```

Tell the machine to take only 9 decimal places seriously by typing 9 and pressing `[RND]`. The result is probably what the machine meant all the time.

```
1: [[ [-2 1 ]
      [ 1.5 -.5 ] ]]
GRAPH NUM.MN GEOM CHLC SEXIE PPRMT
```

### Rounding: Number of Significant Digits

The command RND will not work on numbers that must be presented in scientific notation because they are too large or too small to be displayed otherwise. Also, it will not give the expected result if your calculator is in SCI or ENG mode. The following program, called SIG.D, enables you to round the number in level 1 to a specified number of significant digits, thus changing the precision without having to change the display mode. The program is

```

« → N « XPON LASTARG MANT N 1 - RND SWAP ALOG * » »
checksum: # 28637d

```

Type in this program and enter it onto the stack. Then type 'S I G . D and press **[STO]**.

For example, if the number 1.23456789E20 is in level 1,

```

1: 1.23456789E20
GRAPH NUMN GEOM CHLO REHE PPRM

```

the command 3 **[SIG.D]** rounds it to 3 significant digits.

```

1: 1.23E20
GRAPH NUMN GEOM CHLO REHE PPRM

```

A remark or two is in order about the entering of programs. One may type in the symbols that make up the program as it is written, or one may select the commands from the various menus. In every case, it is necessary to observe the spacing shown and to distinguish between the numeral 0 and the letter capital O. Spacing is crucial for commands such as  $\rightarrow N$  and  $\rightarrow LIST$ , which involve the arrow and letters with or without space between them. If you type the symbols one at a time, there will sometimes be spaces where you don't want them. To get  $\rightarrow LIST$ , it may be easier to go to the PRG OBJ menu and press the key **[ $\rightarrow LIST$ ]**. The same is true for such commands as  $C \rightarrow R$  (found on the PRG OBJ menu) and  $STO+$  (found on the *right-shifted* MEMORY menu).

Another device that aids in correctly typing a program is the checksum, described on p. 101 of the *Manual*. This is a binary integer uniquely associated to the contents of the program. To determine whether you have typed in the program correctly, after it is stored, put the name of the program on the stack and press **[BYTES]** on the MEMORY menu. The checksum and the size of the program are returned; if the checksum is not the same as that given, then there is a typing error in the program. It is assumed that your calculator is in DEC mode, for the checksums are given as "decimal binary" (base ten) integers.

## Decimals to Fractions

The command **[ $\rightarrow Q$ ]** found on the HP 48 is designed to give a "best guess" fraction equivalent to the decimal number on level 1 of the stack. It is not foolproof, but will reliably return fractions with denominators of up to about 5 digits. The *Manual* discusses the  $\rightarrow Q$  command on p. 136, where it indicates that the accuracy of the command depends on the display mode of the calculator. If the display mode is n FIX, then only the first n decimal places are used to guess what the fraction was. If the display mode is STD, then all 11 or 12 decimal places are used.

As an example of the use of  $\rightarrow Q$ , suppose you want to add the fractions  $\frac{3}{8}$  and  $\frac{5}{12}$ . Type the following:

3 **[ENTER]** 8  $\div$  5 **[ENTER]** 12  $\div$

```

2: 375
1: .416666666667
GRAPH NUMN GEOM CHLO REHE PPRM

```



+ 1: .791666666667  
GRAPH NUMN SEOM CALC SEIE PFMCT

$\rightarrow Q$ . 1: '19/24'  
T.M.M. C-00 S.D.S.0 EXCD IOPM S.PM

Or you may do it this way:

'3 ÷ 8 + 5 ÷ 12' ENTER 2: '19/24'  
1: '3/8+5/12'  
T.M.M. C-00 S.D.S.0 EXCD IOPM S.PM

EVAL 2: '19/24'  
1: .791666666667  
T.M.M. C-00 S.D.S.0 EXCD IOPM S.PM

$\rightarrow Q$ . 2: '19/24'  
1: '19/24'  
T.M.M. C-00 S.D.S.0 EXCD IOPM S.PM

The  $\rightarrow Q$  command also works on algebraic expressions, turning the decimal numbers in them into fractions. For example, if the expression

$$.3125X + .428571428571Y = .378666666667$$

is on the stack,

1: '.3125\*X+  
.428571428571\*Y=  
.378666666667'  
GRAPH NUMN SEOM CALC SEIE PFMCT

then pressing  $\rightarrow Q$  produces rational coefficients.

1: '5/16\*X+3/7\*Y=142/  
375'  
GRAPH NUMN SEOM CALC SEIE PFMCT

The command  $\rightarrow Q\pi$  on the ALGEBRA menu takes a decimal number from the stack and expresses it as a fraction times  $\pi$ . It will give spurious results if the number on the stack is not actually a rational multiple of  $\pi$ .

For example, if the number 1.3463985154 is on the stack,

1: 1.3463985154  
GRAPH NUMN SEOM CALC SEIE PFMCT

pressing  $\rightarrow Q\pi$  will return the number '3/7\* $\pi$ '.

1: '3/7\*\pi'  
T.M.M. C-00 S.D.S.0 EXCD IOPM S.PM

(Note that this is read as  $\frac{3}{7}\pi$  or  $\frac{3\pi}{7}$ , not  $\frac{3}{7\pi}$ .)

There are some limitations to the  $\rightarrow Q$  command that should be discussed. If a decimal number corresponds to a fraction with  $k$  digits in the denominator, it may take a mode setting of as much as  $(2k - 1)$  FIX to recover the fraction. It is therefore clear that with a denominator of six or more digits, you should not expect to be able to recover the fraction at all.

One response to this limitation is to use STD mode all the time, but that may be too much accuracy. Here is a simple example of what can go wrong.

It is a fact that  $\frac{5}{12} - \frac{2}{9} = \frac{7}{36}$ . If you type '5/12-2/9' ENTER EVAL →Q while in STD mode, the machine returns  $\frac{8333333331}{42857142845}$ , which you suspect right away is not the right answer. If you are in FIX mode of 10 or less, however, the machine gives the correct result. What has happened is that in computing the result, the machine suffered a roundoff error. The decimal equivalent of  $\frac{7}{36}$  is .194444444444, while the result of the computation is .194444444445. That last digit, the result of a round-off error, spoils the result in STD mode.

It is therefore evident that the most appropriate mode for use of the →Q command is probably 9 FIX. Unfortunately, it is much more convenient to operate in STD mode or something like 4 FIX mode or a SCI mode. To avoid having to change from the present mode in order to use →Q appropriately, the following little program, called D→Q, can be used.

```
« RCLF 9 FIX SWAP →Q SWAP STOF »
checksum: # 23425d
```

Store this program by typing ' ', holding down the α key and typing D → Q, and then releasing α and pressing STO.

This program remembers the mode the machine is in, changes to 9 FIX to apply →Q, and then returns to the previous mode. With this program stored at the HOME level of the VAR menu, it is available for use at all times. (I have assigned the program « D→Q » to a single key on the USER keyboard, so that I don't always have to find the command on the menu.)

## Prime Factorization

The following program, called PFACT, gives the prime factorization of the integer in level 1 on the stack. If the integer is already prime, the number itself is returned. For the symbols not appearing on the keyboard, see p. 52 of the *Manual*.

```
« DUP 'N' STO ( ) 1 CF
  WHILE 1 FC?
    REPEAT N J IP 2 → R K
      «
        WHILE K R ≤ N K MOD 0 ≠ AND
          REPEAT K 1 + 'K' STO
            END
          IF K R >
            THEN N + 1 SF
            ELSE K + N K / 'N' STO
            END
        »
    »
```

```

»
END 'N' PURGE 1 CF
»

```

checksum: # 14205d

Store the program by typing 'P F A C T' STO. As an example of the use of PFACT, suppose you had to reduce the radical  $\sqrt{675}$ . Type

```

675
1:
675
GRAPH NUMM GEOM CALC REGE PFACT

PFACT
2:
1: { 3 3 3 5 5 } 675
GRAPH NUMM GEOM CALC REGE PFACT

```

to get the prime factorization, which is  $3*3*3*5*5$ . The radical can therefore be reduced by pulling out two factors each of 3 and 5, so that  $\sqrt{675} = 15\sqrt{3}$ .



## Chapter 2. Use in Precalculus Topics

Since the calculus constantly refers to precalculus mathematics, it only makes sense that you should learn to handle precalculus mathematics on the HP calculator in order to do calculus on it. In this chapter we take a brief look at some of the uses of the HP 48 in precalculus topics.

### Algebraic Manipulation

The symbol-manipulating calculators can do quite a bit of algebraic manipulation. Some examples of what can be done are shown starting on p. 125 of the *Manual*. The algebraic manipulation commands are explained in detail starting on p. 386.

There are some limitations of which you should be aware. Although the calculator will expand and collect by itself, it will not factor by itself. You can walk it through to get simple factoring done, but you must do it "by hand". If you use the program EXCO from the *Manual*, p. 568, you are sometimes surprised at what the calculator thinks is simplified form.

### Solving Equations

Solving equations is something the calculator can do pretty well. It can do it in either algebraic form, for some equations, or numerical form, for others.

#### Isolating a Variable

The command **ISOL** on the ALGEBRA menu can be used to isolate a single occurrence of a variable in an equation. To use it, enter the equation onto the stack, and then enter the variable you wish to isolate. Then press **ISOL**.

For example, to solve for x in the equation  $x^2 + y^2 = z^2$ , type the following:

```
'X [y^x] 2 + Y [y^x] 2 = Z [y^x] 2 [ENTER] 'X 1:      'X^2+Y^2=Z^2'
                                1: X
                                SOLV EXPN ISOL QUAD SHOW TMLA
[ISOL]                          1:      'X=s1*J(Z^2-Y^2)'
                                1: X
                                SOLV EXPN ISOL QUAD SHOW TMLA
```

The symbol s1 in the result stands for  $\pm 1$ . That is, the machine gives both solutions to a quadratic equation. More generally, the machine will use De Moivre's theorem to give you all n solutions of an n<sup>th</sup>-degree equation. The symbol n1 in a result stands for one of the integers 0, 1, 2, ..., n - 1.

For a quadratic equation, the command **QUAD** on the SOLV menu also isolates the variable, using the quadratic formula. If there is no other variable in the equation, the result is a numerical answer. If there are other variables, the result is in terms of the other variables.



For example, to solve the equation  $x^2 + 3x + 4 = 0$ , type in

'X [y<sup>x</sup>] 2 + 3 \* X + 4 [ENTER] 'X  
 1: 'X^2+3\*X+4'  
 [SOLVE] [EXP] [SOL] [FUNC] [SHOW] [TMYL]  
 [QUAD].  
 1: 'X=(-3+±1\*(0,2.64575131106))/'  
 2:  
 [SOLVE] [EXP] [SOL] [FUNC] [SHOW] [TMYL]

The result in this case is given as a pair (s1 = ±1) of complex numbers.

## Root Finding

There are at least two methods of finding numerical solutions to an equation in a single variable on the HP 48. For a quadratic equation, [QUAD] returns a numerical solution, modulo the symbol s1. Suggestions for further handling quadratic equations are given in the *Manual*, starting on p. 391.

For other equations, the [ROOT] command on the SOLVE menu is used. First enter the equation, then the variable, and then an approximation to the desired root. [ROOT] will find the closest solution to the approximation given.

For example, to find the positive zero of the function  $x^3 + 4x^2 - 3x - 12$ , type in

'X [y<sup>x</sup>] 3 + 4 \* X [y<sup>x</sup>] 2 - 3 \* X - 12  
 [ENTER] 'X [ENTER] 2  
 2: 'X^3+4\*X^2-3\*X-12'  
 1:  
 2:  
 [SOLVE] [ROOT] [NEW] [EQN] [STEP] [CMT]  
 [ROOT].  
 1: 1.73205080757  
 [SOLVE] [ROOT] [NEW] [EQN] [STEP] [CMT]

The Solver offers another way to find roots; we will discuss it next in another context. Other methods of root finding are possible using programs, and will be discussed in the appropriate contexts later.

## Evaluating Expressions

It is often desired to evaluate an expression involving variables for given values of the variables. The HP Solver environment was created precisely to do that easily. The Solver creates a menu for the expression and for each variable in it, enabling you to easily "plug in" values and get values out.

For example, suppose that you wish to evaluate the expression  $\sqrt{a^2 + b^2}$  for various values of a and b. First type the expression

'√ ( A [y<sup>x</sup>] 2 + B [y<sup>x</sup>] 2 [ENTER].  
 1: '√(A^2+B^2)'  
 [GRAPH] [NUM] [EQN] [CMT] [SERIE] [PRMT]

Then go to the SOLV menu and store the expression as your equation by pressing

**STEQ**.

1:  
SOLVR ADD NEW EQN: STY: CNT

This command stores the expression in a container named EQ on the VAR menu. Then enter the Solver environment by pressing

**SOLVR**.

2:  
1:  
A B EXPR=

You will see a special menu with choices labeled **A**, **B**, and **EXPR=**. To set A equal to 4, say, and B equal to 5, type

4 **A** 5 **B**. Then press **EXPR=** to see the value of the expression.

1: EXPR: 6.40312423743  
A B EXPR=

To evaluate at A = 4 and B = 3, just type 3 **B** and press **EXPR=** again.

2: EXPR: 6.40312423743  
1: EXPR: 5  
A B EXPR=

The value of A remains 4 until you change it.

The Solver can also be used to solve for one variable in a formula when the other variables have values given. This is discussed in the *Manual* starting on p. 250. To find a zero of a function  $f(x)$ , put the function  $f$  in the solver, put an approximate value of the zero in the variable X, and then solve for X by left-shifting and then pressing **X**.

Another way to evaluate an expression is to do by hand what the Solver does for you. First, create a container for each variable in the expression by storing a desired value in each. Then type in the expression, and use **EVAL**. This procedure is usually used within a program, where the Solver environment is not available.

For example, to evaluate the function  $f(x) = \frac{\sin x}{x}$  at  $x = 0.15$ , type

. 1 5 **ENTER** ' X

1: .15  
X  
GRAPH NUMN GEOM CMLC SERIE PERMT

**STO**

1:  
X GRAPH NUMN GEOM CMLC SERIE

' S I N ( X **ENTER** ' X **ENTER**

2: 'SIN(X)'  
1: X  
X GRAPH NUMN GEOM CMLC SERIE

**+**

1: 'SIN(X)/X'  
X GRAPH NUMN GEOM CMLC SERIE

**EVAL**.

1: .996254216493  
X GRAPH NUMN GEOM CMLC SERIE

(Make sure the calculator is in radian mode!)

After evaluating an expression, the containers for the variables are left on the VAR menu. It is a good practice to PURGE such containers if they are not going to be used again immediately. Not only it is good housekeeping, but some function commands don't work as you expect when there is a container named for the variable on the VAR menu.

Here is a way to clean up after yourself automatically. The following program, SV, sets up to use the Solver. It creates a subdirectory to hold the equation and the variables used in the Solver, enters that subdirectory, stores the equation on level 1, and then calls the SOLVER menu; it also creates a QUIT button on the subdirectory which purges the subdirectory and everything in it.

```
« 'S' CRDIR S STEQ « UPDIR 'S' PGDIR » 'QUIT' STO 30 MENU »
checksum: # 58764d
```

If you decide to use the Solver, you can fetch or type in your equation, use SV, and solve to your heart's content. When you are done, press **VAR** and then **QUIT**; no trace of using the Solver remains except for the results you left on the stack. If you wish to recall the equation used before using QUIT, just press **EQ** first.

If the program SV resides at the HOME level of memory, then it can be called by typing

SV **ENTER**,

no matter what level of user memory is current. Exiting by pressing **QUIT** leaves the memory and the current level of memory unchanged.

## Graphs of Functions

The HP 48 has many built-in graphics commands that make for flexible and convenient plotting of the graphs of functions. Most of these are accessed through commands on the PLOT menu.

The display on the HP 48 is 131 pixels wide and 64 pixels high. The display can be thought of as a window showing a portion of the Cartesian plane. The particular portion shown, and therefore its scale, is controlled by the contents of PPAR, the plot parameters.

### Plot Parameters

PPAR contains a list, of the form

$$\{ (x_{\min}, y_{\min}) (x_{\max}, y_{\max}) \text{ indep } n (x_{\text{axis}}, y_{\text{axis}}) \text{ plotype dep} \}.$$

The point  $(x_{\min}, y_{\min})$  is at the lower left corner of the window, and the point  $(x_{\max}, y_{\max})$  is at the upper right corner. "indep" is the independent variable used, X in most simple cases. The number  $n$  is a resolution number, and tells the machine whether you want a pixel turned on in each column of the display. The point  $(x_{\text{axis}}, y_{\text{axis}})$  is the point at which the axes intersect, and may or may not be visible through the window. If an axis is shown, it

will have a tic mark every ten pixels. "plotype" identifies the choice made in PTYPE, and tells which type of graph is to be drawn. "dep" is the dependent variable, Y in most cases.

The default plot parameters are

{ (-6.5, 3.1) (6.5, 3.2) X 0 (0, 0) FUNCTION Y }.

These parameters put the axes through the center of the screen, with a tic mark every unit. The container PPAR is on the VAR menu; if no such container is there, entering the PLOT menu creates one with the default plot parameters in it.

The contents of PPAR can be edited if you want to change them, or you may use the built-in commands on the PLOT menu. Some of the contents of PPAR are displayed when you enter the PLOT menu, and others when you enter the PLOTR menu. The points ( $x_{\min}$ ,  $y_{\min}$ ) and ( $x_{\max}$ ,  $y_{\max}$ ) are changed using **XRNG** and **YRNG**. The command **INDEP** is used for changing the independent variable, **DEPN**, for changing the dependent variable, and **RES**, for changing the resolution number. The command **CENT** adjusts the plot parameters so as to move the window, without changing its length or width, so that the point that is on the stack is shown at the center of the screen. The command **SCALE** multiplies the height and width of the window by the two numbers on the stack, relative to the default window. **RESET** resets the default window parameters and erases any previous picture. Other commands are discussed in the *Manual*, beginning on p. 291.

## Drawing

The basic procedure for getting the graph of a function is to type in the function, store it in EQ, and DRAW it. To store the function in EQ, enter the PLOT menu and press **STEQ**. That creates a container named EQ on the VAR menu, if one is not already there, and stores the function in it. The command **EDEQ** places the function in edit mode if you want to change the function slightly; the contents of EQ are shown automatically upon entering the PLOT menu, and again when entering PLOTR. The **DRAW** command on the PLOTR menu puts the graph into the display. If you do not first press **ERASE**, the graph is drawn over whatever is already there; you may create multiple graphs by refraining from using **ERASE**.

Now let's try an example. Suppose you want to draw the graph of the line  $y = \frac{1}{4}x$ . Type

'X ÷ 4 **ENTER**,

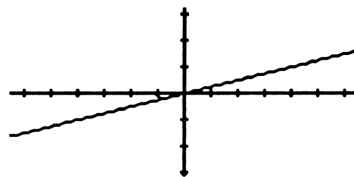
and then press **STEQ** to store that expression,

```

1: 'X/4'
PLOT PTYPE NEW EDEQ STEQ CMY
-----
Indep: 'X'
x:    -6.5    6.5
y:    -3.1    3.2
ERASE DRAW AUTO WAND WAND INDEP

```

and finally press **DRAW**.



Assuming default plot parameters, you will get a nice picture of the line slanting across the screen, passing through the origin as it should.

Press **ON** (actually **ATTN**) to get the stack display back again. Entering the GRAPH menu recalls the current contents of the display screen, so you can see the picture again, and even interact with it.

You should now experiment with various functions and plot parameters, to see how the machine behaves, and how the window works.

If you store an equation such as 'SIN(X) = COS(X)' in EQ, the **DRAW** command plots the graphs of the left-hand side and of the right-hand side simultaneously; this is another way to get two graphs on the screen at the same time.

You may also use a procedure instead of an expression for the function. Suppose you want to create the graph of the function

$$f(x) = \begin{cases} \sin x & \text{if } x < 0 \\ 1 - x^2 & \text{if } x \geq 0 \end{cases}$$

Type « IF X 0 < THEN 'SIN(X)'  
ELSE '1 - X^2' END **ENTER**

```
1: « IF X 0 < THEN '
  SIN(X)' ELSE '1-X^2
  ' END »
PLOT: PTYPE: NEX: EGE: STE: CHY
```

and press **STEQ** and then **DRAW**.



## Saving

Graph storing is automatic on the HP 48; graphs are drawn in a portion of memory called PICT, which does not change until you command a change. For example, a new graph is drawn on top of the previous one unless you ERASE the old one first. The contents of PICT can be stored as a graphics object (another data type) by pressing **PICT** on the PRG DSPL menu and then pressing **RCL**. This graphics object can be viewed again by using the **→LCD** command on the PRG DSPL menu. Also, portions of the display can be selected by setting a mark and referencing the rectangle of which the mark and the cursor are opposite corners; this is described on p. 302 and pp. 337ff of the *Manual*.

### Zooming

Interactive commands for zooming in or out on a picture are built into the ZOOM submenu of the GRAPH menu, which shows automatically after a graph is plotted. Its use is largely self-evident and intuitive, and it is described starting on p. 301 of the *Manual*.

### Multiple Graphs

We have already mentioned two ways to get multiple graphs--by failing to erase a previous graph when plotting a new one, and by storing an equation as the function (the two sides of the equation are plotted separately). If you wish to interact with several graphs on the screen at once (zooming, for instance), then the graphs are plotted by making a list of them. If the list is stored in EQ, then DRAW draws the functions in the order in which they appear in the list.

Graphics objects can be superimposed by just adding (+) them on the stack. This is a method that can be used in a program that creates graphics objects.

### Zeros and Other Graphical Properties

With the graph of a function on the display, a zero of the function is the x-coordinate of a point at which the graph crosses the x-axis; such a point is also called an x-intercept of the graph. A zero can therefore be estimated by noting the x-coordinate of an x-intercept. This can be done by positioning the cursor on the x-intercept and reading the coordinates by pressing **COORD**.

Better yet, press **FCN** and then **ROOT** to get the "exact" value of the zero. Other commands on the FCN submenu allow you to find extrema and function values, the point of intersection of two graphs, and slopes and areas. See pp. 307ff of the *Manual*.

### Plotting Unusual Functions

Occasionally functions occur whose formulas cannot be specified in purely algebraic terms. Such functions include the greatest integer function, the absolute value function, and perhaps some others. Idiosyncrasies of the calculator make it difficult to plot graphs of some other functions, such as those involving fractional exponents. Yet other functions are defined by different formulas for different parts of their domains. Here we look at some ways to work around the difficulties imposed by these types of functions.

#### ABS and FLOOR

The absolute value function,  $f(x) = |x|$ , is known to the calculator as ABS(X). ABS is found on the MTH PARTS menu. For example, to enter the function  $f(x) = |2x - 5|$ , type 'ABS(2\*X-5)'. You can also type '2\*X-5' **ENTER** and then press **ABS**.

The greatest integer function,  $f(x) = \lfloor x \rfloor$ , is known to the calculator as the FLOOR function, found on the MTH PARTS menu. For example, to enter the function  $f(x) = 3\lfloor x - 1 \rfloor$ , type



'3\*FLOOR(X-1)'. You may also type 3 ENTER 'X-1' ENTER and then press FLOOR and \*.

### Fractional Exponents

If the calculator raises a negative number to a fractional power, it automatically puts the result in complex number form, even when the result is a real number. That means that portions of a graph will not be shown when the fractional power of a negative number is involved.

To get the machine to show the entire graph, we use the ABS function in conjunction with the SIGN function, found on the same menu. The SIGN function is defined by

$$\text{SIGN}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}.$$

For example, to enter the function  $f(x) = (x - 2)^{1/3}$ , we write the product

$$\text{'SIGN}(X-2)*(\text{ABS}(X-2))^{(1/3)}.$$

This works, because no negative numbers appear to fractional powers, and the cube root of a negative number is the negative of the cube root of its absolute value.

This same function can also be written using the  $\sqrt[x]{y}$  key, and if  $x$  is odd, the calculator will return values for negative  $y$ . Type 'X-2' ENTER 3  $\sqrt[x]{y}$ , and the result looks like 'XROOT(3,X-2)'.

### Routines as Functions

Sometimes functions are given in "installments," or are defined by different formulas for different parts of their domains. For example, the function

$$f(x) = \begin{cases} \sin x & \text{if } x < 0 \\ 1 - \frac{1}{4}x^2 & \text{if } x \geq 0 \end{cases}$$

can be typed as 'IFTE(X<0,SIN(X),1-X^2/4)'. It can also be graphed by any program that uses DRAW by using the procedure

« IF X 0 < THEN 'SIN(X)' ELSE '1-X^2/4' END »

as the function. As another example, the function

$$f(x) = \begin{cases} 2x & \text{if } x \leq -1 \\ x^2 - 1 & \text{if } -1 < x \leq 2 \\ 1 - x & \text{if } x > 2 \end{cases}$$

can be represented by the procedure

```
«
  IF X -1 ≤
  THEN '2*X'
  ELSE
    IF -1 X < X 2 ≤ AND
    THEN 'X^2-1'
    ELSE '1-X'
    END
  END
»
```

### Animation

One of the outstanding features of the HP 48 is its ability to handle graphics objects quickly, creating animation. You can create your own motion pictures that illustrate situations and concepts much more clearly than still pictures.

Here is a program that takes a list of previously-created graphics objects from the stack and shows them one after another in an endless loop. If the graphics objects were created appropriately, an animation is the result. The program runs until you strike any key. If you strike the ON key, a pile of graphics objects is left on the stack; striking any other key removes everything from the stack.

```
« ERASE ( # 0d # 0d ) PVIEW OBJ→ → N
  «
    DO DUP PICT ( # 0d # 0d ) ROT REPL N ROLL
    UNTIL KEY
    END DROP N
  » DROPN
»
```

checksum: # 55895d

### Synthetic Division

One of the major tools for finding zeros of polynomials is synthetic division. Here is a program that takes a list of the coefficients of a polynomial and a multiplier from the stack and returns the quotient and remainder. I call it SYND.

```
« → L M
  « L L SIZE 0 → N S
    « 1 N
      FOR K L K GET S + DUP M * 'S' STO
      NEXT → R
```

```

« N 1 - →LIST M SWAP R
»
»
»
»
»

```

checksum: # 30901d

For example, to perform the synthetic division of  $x^2 + 4x - 3$  by  $x - 2$ , the list of coefficients is { 1 4 -3 } and the multiplier is 2. Therefore type

```

{ 1 , 4 , 3 [CHS] [ENTER] 2
1:          ( 1 4 -3 )
2:          [SYND] [POL] [RTN] [PRT] [NEXT] [NITE]
[SYND].
4:          ( 1 4 -3 )
3:          2
2:          ( 1 6 )
1:          9
[SYND] [POL] [RTN] [PRT] [NEXT] [NITE]

```

The original data as well as the quotient and remainder are returned to the stack.

## Trigonometric Functions

The trigonometric functions are found on the keyboard of the HP 48. Only the functions SIN, COS, and TAN are given there; the others are found by using the fundamental identities,

$$\cot x = \frac{1}{\tan x}, \quad \sec x = \frac{1}{\cos x}, \quad \text{and} \quad \csc x = \frac{1}{\sin x}.$$

### Angle Modes

The calculator usually comes set in degree mode, meaning that arguments of trigonometric functions are assumed to be in degrees. To change to radian mode, and have the machine treat arguments of trig functions as radians, press [RAD]. The annunciator RAD will come on at the top of the display to signal that you are in radian mode.

It is important to be in radian mode for the applications of calculus, including graphs involving the trig functions. To see a demonstration of that fact, try plotting the function  $\sin x$  in degree mode, and then plot it in radian mode. Type ' S I N ( X [ENTER] to enter the sine function, press [STEQ], and then [DRAW]. It is a good idea to set your machine in radian mode now, and leave it there.

$\pi$

In dealing with radians, the number  $\pi$  will come up a lot. On the HP 48,  $\pi$  and some other numbers are treated as symbolic constants instead of numerical constants. For example, when you press the  $\pi$  key and enter it, the expression ' $\pi$ ' appears on the stack. If you

divide by 6, the expression ' $\pi/6$ ' appears on the stack. Then if you press **SIN**, the expression ' $\text{SIN}(\pi/6)$ ' appears. To change any of these symbolic expressions into a number, press **→NUM**. If the calculator is not in SYM mode, the numerical approximation to  $\pi$  appears instead of ' $\pi$ '.

## Values

Finding values of the trig functions is as simple as pushing the buttons. To find the sine of 1.2 radians, type

1.2  
**SIN**.  
 1: 1.2  
 GRAPH NUM.N GEOM CALC REE T.N.M  
 1: .932039085967  
 GRAPH NUM.N GEOM CALC REE T.N.M

Similar sequences are used to find the cosine and tangent function values.

To find the secant of 1.2 radians, type

1.2 **COS**  
 1: .362357754477  
 GRAPH NUM.N GEOM CALC REE T.N.M  
 and then press the **1/x** key.  
 1: 2.75970360133  
 GRAPH NUM.N GEOM CALC REE T.N.M

Values of the cosecant and cotangent function are found similarly.

## Inverse Trig Functions

Values of the inverse trig functions arcsine, arccosine, and arctangent are found by pressing the buttons **ASIN**, **ACOS**, and **ATAN**. The arcsine and arctangent functions always return a number in the range from  $-\pi/2$  to  $\pi/2$ , and the arccosine function always returns a value between 0 and  $\pi$ . These are the principal value ranges.

If you want values of the arccotangent, use the formula

$$\text{arccot } x = \begin{cases} \arctan \frac{1}{x} & \text{if } x > 0 \\ \frac{\pi}{2} + \arctan \frac{1}{x} & \text{if } x < 0 \end{cases}$$

For example, to find the arccotangent of 2.5, type

2.5,  
 1: 2.5  
 GRAPH NUM.N GEOM CALC REE T.N.M  
 press the **1/x** key,  
 1: .4  
 GRAPH NUM.N GEOM CALC REE T.N.M

and then press **ATAN**.

1: .380506377112  
GRAPH NUM/MN GEOM1 CMLC SERIES T.N.M

To find the arccotangent of -1.5, type

1.5 **+/-** **1/x**

2: .380506377112  
1: -.666666666667  
GRAPH NUM/MN GEOM1 CMLC SERIES T.N.M

**ATAN**

2: .380506377112  
1: -.588002603548  
GRAPH NUM/MN GEOM1 CMLC SERIES T.N.M

$\pi$  2 **÷** **+**

2: .380506377112  
1: |-.588002603548+ $\pi$ /2  
GRAPH NUM/MN GEOM1 CMLC SERIES T.N.M

**→NUM**.

2: .380506377112  
1: .982793723252  
GRAPH NUM/MN GEOM1 CMLC SERIES T.N.M

These cases must be taken into account in order to get the proper principal value range.

For values of the arcsecant function, use the formula

$$\operatorname{arcsec} x = \begin{cases} \arccos \frac{1}{x} & \text{if } x > 0 \\ -\arccos \frac{1}{x} & \text{if } x < 0 \end{cases}$$

For values of the arccosecant function, use the formula

$$\operatorname{arccsc} x = \begin{cases} \arcsin \frac{1}{x} & \text{if } x > 0 \\ -\pi - \arcsin \frac{1}{x} & \text{if } x < 0 \end{cases}$$

It may be that the number on the stack is the sine, cosine, or tangent of a rational multiple of  $\pi$ . The program « ASIN →Q $\pi$  SIN », which I call →SQ, will take the decimal number on the stack and express it as the sine of a rational multiple of  $\pi$ . The program « ACOS →Q $\pi$  COS », called →CQ, will express it as the cosine of a rational multiple of  $\pi$ . The number on the stack must be between -1 and 1 in order for these programs to work. The program « ATAN →Q $\pi$  TAN », called →TQ, will express the decimal number on the stack as the tangent of a rational multiple of  $\pi$ . If the number on the stack is not actually the sine, cosine, or tangent of a rational multiple of  $\pi$ , you will get meaningless results. Your calculator must also be in radian mode for these programs to be valid.

## Applications of Trigonometry

Formulas such as the Law of Sines and the Law of Cosines can be placed into the Solver for efficient handling in the solution of triangles and other applications of the trigonometric functions. Vectors, another topic that often comes up in trigonometry, can be handled directly, as we discuss below under the heading of Linear Algebra.

The solution of triangles can also be automated. Here are four programs that handle the different cases:

ASA: given two angles and the included side

SAS: given two sides and the included angle

SSS: given the three sides

SSA: given two sides and the angle opposite one of them (this is the only ambiguous case, and there may be one solution, two solutions, or no solution at all)

It is recommended that these programs be placed in a TSOL (for Triangle SOLUTIONS) subdirectory, whose menu would look like

ASA      SAS      SSS      SSA

SSA is the program

```
« → a b A
  « b A SIN * →H
    « a H
      IF <
        THEN "No Triangle"
      ELSE 'a(given)' a = 'b(given)' b = 'A(given)' A =
-17
      IF FS?
        THEN π →NUM
        ELSE 180
      END → S
      « a H
        IF ==
          THEN S 2 / 'B' SWAP = S 2 / A - 'C' SWAP =
'J(b^2-a^2)' EVAL 'c' SWAP =
          ELSE a b
            IF ≥
              THEN b A SIN * a / ASIN DUP 'B' SWAP = SWAP
A + S SWAP - DUP 'C' SWAP = SWAP SIN a * A SIN / 'c' SWAP =
              ELSE b A SIN * a / ASIN → B
                « B S B - R→C 'B' SWAP = S A B + - B A -
→ C C1
```

```

      « C C1 R→C 'C' SWAP = a C SIN * A SIN
/ a C1 SIN * A SIN / R→C 'c' SWAP =
    »
  »
END
END
»
END
»
»
»
»

```

For example, given the sides  $a = 1$ ,  $b = 2$ , and angle  $A = 30^\circ$ , put the calculator into degree mode and type

```

1 [ENTER] 2 [ENTER] 30
2:
1:
30
M:M  S:M  C:C  C:M  :
4:      'A(given)=30'
3:      'B=90'
2:      'C=60'
1:      'c=1.73205080757'
M:M  S:M  C:C  C:M  :

```



5 [ENTER] 7 [ENTER] 8 [SSS].

```

4:      'C=8'
3:      'A=38.2132107018'
2:      'B=60'
1:      'C=81.7867892983'
M.M  SW  SD  SD

```

SAS is the program

```

« → a C b
«
  IF a b >
  THEN b a 'b' STO 'a' STO
  END 'a(given)' a = 'C(given)' C = 'b(given)' b =
  '√(a^2+b^2-2*a*b*COS(C))' EVAL → c
  « a c / C SIN * ASIN DUP 'A' SWAP = SWAP C + -17
  IF FS?
  THEN π →NUM
  ELSE 180
  END SWAP - 'B' SWAP = 'c' c =
  »
»
»
»

```

checksum: # 46305d

To use this program, enter the sides and the included angle in the order S, A, S. Then press [SAS]. Again, be consistent in the angle mode used. For example, if the two sides are 1 and 2 and the included angle is 60°, put the calculator into degree mode and type

1 [ENTER] 60 [ENTER] 2 [SAS].

```

4:      'b(given)=2'
3:      'A=29.9999999999'
2:      'B=90.0000000001'
1:      'c=1.73205080757'
M.M  SW  SD  SD

```

ASA is the program

```

« → A c B
« 'A(given)' A = 'c(given)' c = 'B(given)' B = -17
  IF FS?
  THEN π →NUM
  ELSE 180
  END A B + - → C
  « 'C' C = A SIN c * C SIN / 'a' SWAP = B SIN c * C SIN
  / 'b' SWAP =
  »
»
»

```

»

checksum: # 13387d

To use this program, enter the angles and the included side in the order A, S, A. Be consistent with the angle mode. For example, if the angles given are 30° and 60° and the included side is 4, then type

30 [ENTER] 4 [ENTER] 60 [ASA].

```
4: 'B(given)=60'
3: 'C=90'
2: 'a=2'
1: 'b=3.46410161514'
MODE MODE MODE MODE
```

## Logarithmic and Exponential Functions

The [LOG] command returns the common logarithm of the number in level 1 on the stack.

That is, if x is on the stack, [LOG] returns  $\log_{10} x$ . [10<sup>x</sup>] is the common antilogarithm. [LN]

is the natural logarithm, or logarithm base e, and [e<sup>x</sup>] is the natural exponential function.

Logarithms with any positive base  $b \neq 1$  can be found from the formula

$$\log_b x = \frac{\log x}{\log b}.$$

For example, to find  $\log_2 5$ , type

5 [LOG] 2 [LOG]

```
2: .698970004336
1: .301029995664
GRAPH NUM.MN GEOM1 CHLC SERIE T.M.M
```

÷.

```
1: 2.32192809489
GRAPH NUM.MN GEOM1 CHLC SERIE T.M.M
```

Exponentiation with any base  $b > 0$  is easily accomplished with the [<sup>y</sup>x] command. For example, to find  $(1.7)^\pi$ , type

1.7 [ENTER] [π] [<sup>y</sup>x]

```
1: '1.7^π'
GRAPH NUM.MN GEOM1 CHLC SERIE T.M.M
```

[→NUM].

```
1: 5.29634953
GRAPH NUM.MN GEOM1 CHLC SERIE T.M.M
```

## Linear Algebra

Linear algebra is the broad area covering systems of linear equations, matrices, determinants, and vectors and vector spaces. At the precalculus level, everything we need to do can be handled very efficiently by the built-in functions of the calculator, with the exception of the Gauss-Jordan reduction technique.

## Vectors

A vector in the HP 48 is a set of numbers (coordinates) enclosed in brackets, [ ]. To enter a vector, start with [ ], and then enter the numbers, typing either a space or a comma between coordinates.

For example, to enter the vector [ 3 4 -1 ], type [ 3 [SPC] 4 [SPC] 1 [+/-] [ENTER] .

Vector addition and subtraction is accomplished using the [ + ] and [ - ] keys, just as with real numbers. However, if the vectors do not have the same length (number of coordinates), you will get an error message.

For example, to find the sum of the two vectors [ 1 2 -2 ] and [-5 3 2], type

[ 1 [SPC] 2 [SPC] 2 [+/-] [ENTER] [ 5 [+/-]  
[SPC] 3 [SPC] 2 [ENTER]  
[ + ] .

2: [ 1 2 -2 ]  
1: [ -5 3 2 ]  
GRAPH NUMN GEOM CHLC SERIE T.M.M  
1: [ -4 5 0 ]  
GRAPH NUMN GEOM CHLC SERIE T.M.M

The scalar product of a number and a vector can be found by using the [ × ] key. Just place the vector and the scalar in levels 1 and 2 (in either order) and multiply.

The number of coordinates of a vector is given by the [SIZE] command on the PRG OBJ menu. The magnitude or norm or absolute value of a vector is given by the [ABS] command on the MTH VECTR menu.

Dot and cross products of vectors are also defined in the machine. The [DOT] command computes the dot product of any two vectors of the same length. The [CROSS] command computes the vector that is the cross product of two vectors of length two or three. These will be quite useful to us later on.

One restriction that is sometimes unhandy is that the entries of a vector must be real numbers or complex numbers. One cannot use symbols as the coordinates of a vector. If you wish to do so, you might try using lists, whose entries can be anything, and create routines to do arithmetic and whatever else you wish with them.

Another way to enter a vector is to place the coordinates of the vector on the stack, place the number of coordinates in level 1, and then use the [→ARRY] command on the PRG OBJ menu. For example, to create the vector [3 5 2 1], you may type

3 [ENTER] 5 [ENTER] 2 [ENTER] 1  
[ENTER] 4  
[→ARRY] .

3: 5  
2: 2  
1: 1  
4  
OBJ→ E→ M→ L→ L→ S→ T→  
1: [ 3 5 2 1 ]  
OBJ→ E→ M→ L→ L→ S→ T→

This is the method most often used inside a program. The commands  $\rightarrow V2$  and  $\rightarrow V3$  on the MTH VECTR menu take two or three numbers from the stack and create vectors of the indicated lengths, as do the commands  $\boxed{2D}$  and  $\boxed{3D}$  on the keyboard.

## Matrices and Determinants

A matrix is perhaps best regarded as a vector of vectors. The MatrixWriter environment described on p. 346 of the *Manual* is made to make entering, editing, and viewing of matrices easy. For example, to enter the matrix  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ , type  $\boxed{\text{MATRIX}}$   $\boxed{1}$   $\boxed{\text{SPC}}$   $\boxed{2}$   $\boxed{\text{ENTER}}$   $\boxed{\nabla}$   $\boxed{3}$   $\boxed{\text{SPC}}$   $\boxed{4}$   $\boxed{\text{ENTER}}$   $\boxed{\text{ENTER}}$ .

Arithmetic with matrices is done using the regular arithmetic keys. As with vectors, the matrices must have the same size (same numbers of rows and of columns) in order to form sums and differences. Matrices can also be multiplied by using the  $\boxed{\times}$  key, but they must be conformable. That is, the number of columns of the matrix in level 2 must be the same as the number of rows of the matrix in level 1.

The product of a scalar and a matrix can be found by using the  $\boxed{\times}$  key. Just place the matrix and the scalar in levels 1 and 2 (in either order) and multiply.

The  $\boxed{\text{SIZE}}$  command, when applied to a matrix, returns a list in which the first element is the number of rows and the second element is the number of columns. If a matrix is square (has the same number of rows as columns), then the  $\boxed{\text{DET}}$  command on the MTH MATR menu returns the determinant of the matrix. If the determinant of a square matrix is not zero, then the inverse of the matrix exists, and the inverse can be found by using the  $\boxed{1/x}$  command.

(Warning: On early versions of the HP-48SX, there may be a bug in the  $\boxed{1/x}$  command that causes trouble when inverting a matrix larger than 7 by 7. For large matrices, put an identity matrix of the same size on level 2 by using  $\boxed{\text{IDN}}$ , put the matrix to be inverted on level 1, and use  $\boxed{+}$ .)

Other commands on the MTH MATR menu are described on p. 359 of the *Manual*.

A matrix can also be entered by placing its entries on the stack, and then using a list of two numbers to specify the size of the matrix and using  $\boxed{\rightarrow \text{ARRY}}$ . For example, to enter the matrix  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ , we can type

1  $\boxed{\text{ENTER}}$  2  $\boxed{\text{ENTER}}$  3  $\boxed{\text{ENTER}}$  4  $\boxed{\text{ENTER}}$   
{ 2 , 2

4: \_\_\_\_\_ 2  
3: \_\_\_\_\_ 3  
2: \_\_\_\_\_ 4  
1: \_\_\_\_\_ { 2 2 }  
DEF → E → M → L → S → T →

→ARRY.

1:  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$   
 GRAPH NUM.M GEOM CALC SERIE T.M.M

This is most often done in the midst of a program.

A matrix is most easily edited by using the MatrixWriter. With the matrix on level 1, press  $\blacktriangledown$ , and use the cursor keys to move to the desired entries and edit them. To save the edited result, press  $\boxed{\text{ENTER}}$ ; to discard the edited version, press  $\boxed{\text{ON}}$ ; in either case, the matrix appears again on level 1.

### Systems of Linear Equations

The built-in matrix functions of the HP 48 enable the solving of systems of linear equations. Each such system can be represented as a matrix equation of the form  $\mathbf{AX} = \mathbf{B}$ , with  $\mathbf{A}$  the coefficient matrix,  $\mathbf{X}$  the column of unknowns, and  $\mathbf{B}$  the column of constants. If  $\mathbf{A}$  is invertible, then  $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$ , and  $\mathbf{A}^{-1}\mathbf{B}$  can be found by placing the vector  $\mathbf{B}$  in level 2, matrix  $\mathbf{A}$  in level 1, and pressing  $\boxed{+}$ . Even if  $\mathbf{A}$  is not a square matrix, the same approach can be used, as is discussed on p. 362 of the *Manual*. The use of  $\boxed{+}$  involves some roundoff error, typically.

### Gauss-Jordan Reduction

The Gauss-Jordan reduction method of solving a system of linear equations begins with the augmented matrix  $[\mathbf{A} \ \mathbf{B}]$  of the system  $\mathbf{AX} = \mathbf{B}$ . Then by using elementary row operations, the matrix is transformed to reduced echelon form, from which the solution of the original system can be easily read.

The three elementary row operations are (1) interchanging two rows, (2) multiplying a row vector by a nonzero scalar, and (3) replacing a row vector by the vector sum of that row and another row. The latter two can be combined and used repeatedly to change the column containing a chosen nonzero element into a column of zeros, except that the chosen element will be replaced by a 1. This process is called pivoting, and the chosen element is called the *pivot*. The Gauss-Jordan process simply pivots on diagonal elements, insofar as that is possible, until the reduced-echelon form of the matrix is obtained.

In the pivoting process, an entry often is not "zeroed out" as desired, but replaced by a very small number, such as  $1.2 \times 10^{-11}$ . This is due to round-off error in the calculator. To overcome the problem, we use the command RND to instruct the calculator how many decimal places to take seriously.

Here is a program called EXRIJ for exchanging rows I and J of a matrix. It assumes that the matrix is in level 3 of the stack, I is in level 2, and J is in level 1. The matrix, with rows I and J interchanged, is returned to level 1.

```
« → I J
  « DUP SIZE 1 GET IDN 'E' STO 0 0 'E(I,I)' STO 'E(J,J)' STO
  1 1 'E(I,J)' STO 'E(J,I)' STO E SWAP * 'E' PURGE
```

»  
»

checksum: # 6505d

For example, to interchange rows 1 and 3 of the matrix on the stack in level 1, type

1 SPC 3

```

RAD
{ HOME NU.AN MTRX }
1: [[ [ 1 2 3 4 ]
      [ 2 3 4 5 ]
      [ 3 4 5 6 ] ]
1 3
GJRE PIVOT EXRIJ M-0:

```

EXRIJ.

```

1: [[ [ 3 4 5 6 ]
      [ 2 3 4 5 ]
      [ 1 2 3 4 ] ]
GJRE PIVOT EXRIJ M-0:

```

The next program is called PIVOT, and is used to pivot on the element in row K and column L. (The specified element cannot be zero.) It assumes the matrix is in level 3, K is in level 2, and L is in level 1. The program is

```

« → A K L
  « A SIZE 1 GET → M
    « M IDN 'P' STO 1 M
      FOR I 'A(I,L)' EVAL 'P(I,K)' STO
        NEXT M IDN P / A * 9 RND 'P' PURGE
    »
  »
»
»
»

```

checksum: # 58129d

For example, to pivot on the 2,2 element of the matrix on the stack, type

2 SPC 2

```

1: [[ [ 3 4 5 6 ]
      [ 2 3 4 5 ]
      [ 1 2 3 4 ] ]
2 2
GJRE PIVOT EXRIJ M-0:

```

PIVOT.

```

1: [[ [ 0.33 0.00 -0.33...
      [ 0.67 1.00 1.33...
      [ -0.33 0.00 0.33... ]
GJRE PIVOT EXRIJ M-0:

```

EXRIJ and PIVOT are enough now to row-reduce a matrix. You can use them to row-reduce a matrix "by hand" by simply pivoting on the diagonal elements of a matrix, interchanging rows when it is necessary because of the appearance of zeros on the diagonal.

The following program, called GJRED, automates the row-reduction process, starting with a matrix and returning both the matrix and the reduced-echelon form. It also uses the largest possible element to pivot on ("partial column pivoting"), thereby reducing roundoff error as much as possible.

```

« 'A' STO A SIZE OBJ→ DROP 'N' STO 'M' STO 1 1 'C' STO 'R'
STO 1 CF
  WHILE 1 FC?
  REPEAT 'A(R,C)' EVAL ABS 'T' STO R 'K' STO
    IF R M <
    THEN 1 R + M
    FOR I
      IF 'A(I,C)' EVAL ABS DUP 'T1' STO T >
      THEN T1 'T' STO I 'K' STO
    END
  NEXT
END
IF T 0 >
THEN
  IF K R ≠
  THEN A R K EXRIJ 'A' STO
  END A R C PIVOT 'A' STO 1 'R' STO+
  IF R M >
  THEN 1 SF
  END
END 1 'C' STO+
IF C N >
THEN 1 SF
END
END A ( T T1 K R C M N A ) PURGE 1 CF
»

```

checksum: # 11498d

The command STO+ is found on the *right-shifted* MEMORY menu.

Often, especially when solving a system of linear equations, you want to see the results of the Gauss-Jordan reduction in fraction form. The last column of the reduced matrix contains the solution, if the solution is unique. If the solution is not unique, then a row-by-row inspection of the reduced matrix is necessary to specify the complete solution. Each entry of the solution could be isolated and have  $\rightarrow Q$  applied to it, but here is a program that allows for an entire matrix to be transformed into fraction form at once. The program is called M $\rightarrow$ Q, and takes the matrix from level 2 and returns lists of the elements, row by row, in fraction form.

```

« → A
  « A SIZE OBJ→ DROP → M N
  « 1 M
    FOR I 1 N
      FOR J 'A(I,J)' EVAL D→Q
    NEXT N →LIST
  »
»

```



NEXT

»  
»  
»

checksum: # 31133d

It should be noted here that the Gauss-Jordan reduction by machine will not always work. One type of problem that arises is that elements are not really "zeroed out" by the machine, because of round-off error. Thus the machine sometimes finds a small number that is not zero in a position where there should be a zero, and if that element is used as a pivot, the outcome is nonsense. That most often takes place when the system is dependent or inconsistent.

Another problem that arises is that of the "ill-conditioned" matrix. If the determinant of a square matrix is very large or very small in comparison with the elements of the matrix, then the machine is going to have difficulty inverting the matrix, whatever method is used. A simple test for square matrices is to construct the "condition number" of the matrix. If the condition number is greater than the number of digits carried by the machine, then the machine will have difficulty. For the HP 48, that means the condition number should be no greater than 11. Without going into the theory, here is a little program, COND, that gives the condition number of the square matrix that is on the stack.

« → A « A A RNRN LOG A INV RNRN LOG + » »

checksum: # 49918d

## Combinatorics

There are several functions built into HP calculators that calculate certain combinatoric numbers. These include the factorials, permutations and combinations. On the HP 48, the commands are  $!$ ,  $\text{COMB}$ , and  $\text{PERM}$ , all on the MTH PROB menu.

To find  $n!$ , put  $n$  on the stack and press  $!$ . If the number  $x$  on the stack is not a positive integer, the calculator returns the gamma function  $\Gamma(x + 1)$ .

To find  $\binom{n}{k}$ , the number of combinations of  $n$  things taken  $k$  at a time, put  $n$  and  $k$  on the stack and press  $\text{COMB}$ . For example, to compute  $\binom{5}{2}$ , type 5  $\text{ENTER}$  2  $\text{COMB}$ .

To find  $P(n,k)$ , the number of permutations of  $n$  things taken  $k$  at a time, put  $n$  and  $k$  on the stack and press  $\text{PERM}$ . For example, to compute  $P(5,2)$ , type 5  $\text{ENTER}$  2  $\text{PERM}$ .

These can be used together to compute probabilities, etc. For example, to compute the probability of a full house in a random draw of five cards from a standard playing deck, which is

$$\frac{13 \cdot \binom{4}{3} \cdot 12 \cdot \binom{4}{2}}{\binom{52}{5}},$$

type 13  $\boxed{\text{ENTER}}$  4  $\boxed{\text{ENTER}}$  3  $\boxed{\text{COMB}}$   $\boxed{\times}$  12  $\boxed{\times}$  4  $\boxed{\text{ENTER}}$  2  $\boxed{\text{COMB}}$   $\boxed{\times}$  52  $\boxed{\text{ENTER}}$  5  $\boxed{\text{COMB}}$   $\boxed{+}$ . Use  $\text{D}\rightarrow\text{Q}$  to see the fraction form.

## Elementary Plane Analytic Geometry

Many of the operations of analytic geometry are easy to do on the HP 48 because of the ability of the calculator to handle points (it thinks it is handling complex numbers) and vectors. Here are some tricks and little programs.

### Distance Formula

To find the distance between points (a, b) and (c, d) in the plane, just find the absolute value of the difference of the complex numbers (a, b) and (c, d).

For example, to find the distance between (5, 3) and (7, -1), type

( 5 , 3 $\boxed{\text{ENTER}}$ ( 7 , 1 $\boxed{+/-}$ $\boxed{\text{ENTER}}$	2: (5,3) 1: (7,-1) $\boxed{\text{GRAPH}} \boxed{\text{NUMN}} \boxed{\text{GEOM}} \boxed{\text{CHLD}} \boxed{\text{SERIE}} \boxed{\text{T.M.M}}$
-	1: (-2,4) $\boxed{\text{GRAPH}} \boxed{\text{NUMN}} \boxed{\text{GEOM}} \boxed{\text{CHLD}} \boxed{\text{SERIE}} \boxed{\text{T.M.M}}$
$\boxed{\text{ABS}}$ .	1: 4.472135955 $\boxed{\text{REC}} \boxed{\text{SIGN}} \boxed{\text{CONJ}} \boxed{\text{ABS}} \boxed{\text{RE}} \boxed{\text{IM}}$

You may type  $\boxed{\text{SPC}}$  instead of the comma.

### Midpoint Formula

To find the midpoint of the segment whose endpoints are (a, b) and (c, d) in the plane, just find the average of the complex numbers (a, b) and (c, d).

For example, to find the midpoint between (-1, 3) and (2, -3), type

( 1 $\boxed{+/-}$ , 3 $\boxed{\text{ENTER}}$ ( 2 , 3 $\boxed{+/-}$ $\boxed{\text{ENTER}}$	2: (-1,3) 1: (2,-3) $\boxed{\text{GRAPH}} \boxed{\text{NUMN}} \boxed{\text{GEOM}} \boxed{\text{CHLD}} \boxed{\text{SERIE}} \boxed{\text{T.M.M}}$
+ 2	1: (1,0) 2: $\boxed{\text{GRAPH}} \boxed{\text{NUMN}} \boxed{\text{GEOM}} \boxed{\text{CHLD}} \boxed{\text{SERIE}} \boxed{\text{T.M.M}}$
$\div$ .	1: (.5,0) $\boxed{\text{GRAPH}} \boxed{\text{NUMN}} \boxed{\text{GEOM}} \boxed{\text{CHLD}} \boxed{\text{SERIE}} \boxed{\text{T.M.M}}$

### The Line on Two Points

This program, called  $\text{PP}\rightarrow\text{L}$ , takes two points (in complex number form) from the stack and returns the equation of the line they determine.

```

« → P1 P2
  « P1 C→R 1 3 →ARRY P2 C→R 1 3 →ARRY CROSS → L
    « L 1 GET 'X' * L 2 GET 'Y' * + L 3 GET + 0 =
      »
    »
  »
»

```

checksum: # 46969d

For example, to get the line on the points (1, 2) and (-1, 1), type

```

( 1 , 2 [ENTER] ( 1 [+/-] , 1 [ENTER]
2: (1,2)
1: (-1,1)
PP→L LL→P COL: CON: G.PTL P3→π
PP→L.
1: 'X-2*Y+3=0'
PP→L LL→P COL: CON: G.PTL P3→π

```

The name of the program reminds you to enter two points, and the output is a line.

### The Point on Two Lines

This program, called LL→P, takes two vectors from the stack, representing two lines in the plane, and returns the point of intersection of the two lines. The line  $ax + by + c = 0$  is entered as the vector  $[a \ b \ c]$ . If the two lines are parallel, the result "PARALLEL" is returned.

```

« CROSS OBJ→ DROP → A B C
  «
    IF C 0 ==
    THEN "PARALLEL"
    ELSE A C / B C / R→C
    END
  »
»

```

checksum: # 27494d

For example, to find the point of intersection of the two lines  $3x + 2y - 1 = 0$  and  $x - 2y + 3 = 0$ , enter the coefficients as vectors:

```

[ 3 , 2 , 1 [+/-] [ENTER] [ 1 , 2
+/-] , 3 [ENTER]
2: [ 3 2 -1 ]
1: [ 1 -2 3 ]
PP→L LL→P COL: CON: G.PTL P3→π
LL→P.
1: (-.5,1.25)
PP→L LL→P COL: CON: G.PTL P3→π

```

The name of the program reminds you to enter two lines, and the result is a point.

## Collinear Points

This program, called COL?, takes three points from the stack and determines whether they are collinear. If so, the line on which they lie is given. Here is the program :

```
« PP→L 'L' STO C→R 'Y' STO 'X' STO L EVAL OBJ→ DROP2
  IF ==
  THEN "YES" L
  ELSE "NO"
  END { L X Y } PURGE
```

»

checksum: # 17469d

For example, to determine whether the points (-1, -3), (1, 4), and (5, 12) are collinear, type

( 1 $\boxed{+/-}$ , 3 $\boxed{+/-}$ $\boxed{\text{ENTER}}$ ( 1 , 4	3:	(-1, -3)
$\boxed{\text{ENTER}}$ ( 5 , 12 $\boxed{\text{ENTER}}$	2:	(1, 4)
	1:	(5, 12)
	$\boxed{\text{PP→L}} \boxed{\text{LL→P}} \boxed{\text{COL?}} \boxed{\text{CON?}} \boxed{\text{OPTL}} \boxed{\text{PE→T}}$	
$\boxed{\text{COL?}}$ .	1:	"NO"
	$\boxed{\text{PP→L}} \boxed{\text{LL→P}} \boxed{\text{COL?}} \boxed{\text{CON?}} \boxed{\text{OPTL}} \boxed{\text{PE→T}}$	

Then try the points (-1, -3), (1, 4), and (5, 18).

## Concurrent Lines

This program, called CON?, takes three vectors, representing three lines in the plane, from the stack and determines whether the lines all pass through the same point. If so, the point of intersection is given. The line  $ax + by + c = 0$  is entered as the vector [ a b c ].

```
« → L1 L2
  « L1 L2 CROSS DOT
    IF 0 ==
    THEN "YES" L1 L2 LL→P
    ELSE "NO"
    END
```

»

»

checksum: # 54109d

For example, to determine whether the lines  $2x - y + 3 = 0$ ,  $x + 4y + 1 = 0$ , and  $3x + 21y + 2 = 0$  are concurrent, type

[ 2 , 1 $\boxed{+/-}$ , 3 $\boxed{\text{ENTER}}$ [ 1 , 4 , 1 $\boxed{\text{ENTER}}$ [	3:	[ 2 -1 3 ]
3 , 21 , 2 $\boxed{\text{ENTER}}$	2:	[ 1 4 1 ]
	1:	[ 3 21 2 ]
	$\boxed{\text{PP→L}} \boxed{\text{LL→P}} \boxed{\text{COL?}} \boxed{\text{CON?}} \boxed{\text{OPTL}} \boxed{\text{PE→T}}$	

[CON?].

```

2: "YES"
1: (-1.444444444444,
   .111111111111)
PP→L LL→P COL→ CON→ D.PTL P3→π

```

If the message "PARALLEL" appears, it means that the three lines are all parallel (and hence meet in a point at infinity).

### Distance from a Point to a Line

This program, called D.PTL, takes a point (a, b) and a vector [ p q r ] from the stack and returns the distance from the point (a, b) to the line  $px + qy + r = 0$ .

```

« → P L
  « P C→R 1 3 →ARRY L DOT ABS L OBJ→ DROP2 2 →ARRY ABS /
  »
»

```

checksum: # 22512d

For example, to find the distance from the point (5, 3) to the line  $3x + 2y - 5 = 0$ , type

( 5 , 3 [ENTER] [ 3 , 2 , 5  
[+/-] [ENTER]

[D.PTL].

```

2: (5,3)
1: [ 3 2 -5 ]
PP→L LL→P COL→ CON→ D.PTL P3→π

1: 4.43760156981
PP→L LL→P COL→ CON→ D.PTL P3→π

```

## **Chapter 3. Limits and Derivatives**

The capabilities of the HP 48 make possible the investigation of functions in ways never seen in a calculator before. The HP 48 can find derivatives of all the elementary functions. Derivatives are presented in unsimplified form, illustrating the various differentiation formulas. Specifying the independent variable is necessary, so that partial differentiation is also possible. That makes implicit differentiation possible, for which a simple program is presented. In this chapter, we mention limits, derivatives and their applications, and function analysis, including the bisection method and Newton's method for estimating zeros of functions. We conclude this chapter with environments for handling function tables and difference tables.

### **Limits**

The evaluation of limits via calculator is sometimes possible, in perhaps a couple of ways. One is graphical, and the other is computational, but these are really the same thing when you think about it.

#### **Graphical Evaluation**

To examine the limit  $\lim_{x \rightarrow c} f(x)$ , try graphing the function  $f$  over an interval containing  $c$ . You may then zoom in toward  $c$  to "blow up" the picture, and arrive at a conclusion about the limit in that way.

#### **Computational Evaluation**

Another way to examine the limit  $\lim_{x \rightarrow c} f(x)$  is by using the Solver with  $f(x)$  as the expression, and evaluate at values successively nearer to  $c$ . You may be able to arrive at a conclusion about the limit in that way.

### **Finding Derivatives**

#### **Zooming**

Graphically, the derivative of a function is the slope of its graph. To see the slope graphically, pick the point on the graph at which you wish to see the derivative and make that point the center of the display by positioning the cursor on that point and pressing CNTR. Then zoom in until the graph appears to be a straight line; the slope of that line is the value of the derivative at that point.

If the graph never appears to be a straight line, no matter how much zooming in you do, then the curve is not smooth there, and the derivative does not exist.

To find the derivative of a function algebraically, enter the function, specify its independent variable, and press  $\boxed{\partial}$ . The derivative is displayed.

For example, to find the derivative of  $f(x) = \cos 2x$ , type

'COS(2\*X'  $\boxed{\text{ENTER}}$  'X

1: 'COS(2\*X)'  
 $\boxed{\partial}$  X'  
 GRAPH NUM MODE GEOM1 CALC SERIES T.M.M

$\boxed{\partial}$ .

1: '-(SIN(2\*X)\*2)'  
 GRAPH NUM MODE GEOM1 CALC SERIES T.M.M

The derivative is presented in the form that illustrates the chain rule. You can also see why being in radian mode is important, if you happened to be in degree mode.

As another example, suppose that from the formula  $V = \pi r^2 h$  you wish to find  $\frac{dV}{dr}$ . Enter the right-hand side of the formula and differentiate, by typing

'PI \* R ^ 2 \* H'  $\boxed{\text{ENTER}}$  'R

1: 'PI\*R^2\*H'  
 $\boxed{\partial}$  R'  
 GRAPH NUM MODE GEOM1 CALC SERIES T.M.M

$\boxed{\partial}$ .

1: 'PI\*(2\*R)\*H'  
 GRAPH NUM MODE GEOM1 CALC SERIES T.M.M

### Simplifying

Since derivatives are given in unsimplified form, you can simplify them somewhat by using  $\boxed{\text{COLCT}}$  and  $\boxed{\text{EXPAN}}$  on the ALGEBRA menu. For many applications, however, a simplified form is not needed, and the calculator has no trouble working with the unsimplified form. It sometimes takes longer, however, in unsimplified form.

### Evaluating

Evaluating a derivative is done in exactly the same ways that other expressions are evaluated. There is one additional means, however, that you should be aware of.

If a function  $f$  uses the variable  $X$ , and a container named  $X$  (containing the number  $c$ ) is on the current level of the VAR menu, then when the  $\boxed{\partial}$  command is given, not only is the derivative computed, but it is evaluated at the number  $c$  contained in  $X$ . The value  $f'(c)$ , not the derivative  $f'$ , is returned. This can be convenient when you want to evaluate the derivative, but it is not so convenient if you really want the function  $f'$ . If you want the function  $f'$ , make sure that no container named  $X$  is on the VAR menu.

Here is a way to make sure that no container named  $X$  will be present—just delete them all! The following program, XPRG, will delete any container named  $X$  in the current path.

```
« PATH → P
  « 1 P SIZE
    FOR K P K GET EVAL 'X' PURGE
```

NEXT

»

checksum: # 62107d

## Higher-order Derivatives

Finding higher-order derivatives is no different on the calculator than it is by hand; we just differentiate again. We must specify the independent variable each time, though.

If you want to automate the finding of higher-order derivatives, you can construct a program that will return the particular higher-order derivative that you want. Here is one that will take the function from level 2 and the order  $n$  of the derivative from level 1 and return the  $n^{\text{th}}$  derivative; this program is called DNDX.

```
« 1 SWAP START 'X' » COLCT NEXT »
```

checksum: # 32234d

For example, if you want the 3<sup>rd</sup> derivative of  $f(x) = x^5 - 4x^3$ , type

```
'X ^ 5 - 4 * X ^ 3' [ENTER] 3
```

```
1: 'X^5-4*X^3'
3
[ONCE] [IMPO] [TLIN] [NOTED] [PCH] [PCH]
```

```
[DNDX].
```

```
1: '-24+60*X^2'
[ONCE] [IMPO] [TLIN] [NOTED] [PCH] [PCH]
```

Note that the program assumes that  $X$  is the variable; the name of the program reminds you of that.

## Implicit Differentiation

The HP 48 can be programmed to produce the derivative of an implicit function. The theory behind it is couched in partial derivatives; the formula is based on the theorem that if  $f(x, y) = 0$ , then  $\frac{dy}{dx} = -\frac{\partial f / \partial y}{\partial f / \partial x}$ . Here is the program, called IMPD, which starts with the expression

$f(x, y)$  on the stack and returns both the expression  $f$  and the derivative  $\frac{dy}{dx}$ .

```
« DUP 'Y' » COLCT 'FY' STO DUP 'X' » NEG COLCT FY / COLCT
'FY' PURGE
»
```

checksum: # 6634d

For example, given  $x^2 + y^3 = 5$ , the derivative is found implicitly by typing

```
'X ^ 2 + Y ^ 3 - 5' [ENTER]
```

```
1: 'X^2+Y^3-5'
[ONCE] [IMPO] [TLIN] [NOTED] [PCH] [PCH]
```



**IMPD**.

```

2:      'X^2+Y^3-5'
1:      '-(.6666666666666666*X*
      Y^-2)'
GN08: IMP0 TLIN ROT0 PCH0 PCH2

```

Try pressing **→Q** to make the result look better.

```

2:      'X^2+Y^3-5'
1:      '-(2/3*X*Y^-2)'
GN08: IMP0 TLIN ROT0 PCH0 PCH2

```

## Tangent Lines

The simplest application of the derivative of a function is to find the tangent line to the graph of the function at a specified point. For example, if  $y = f(x)$  is given, then the tangent line to the graph at  $(c, f(c))$  is  $y = f(c) + f'(c)(x - c)$ . The derivative  $f'(c)$  is easily found, as described above, and a program can be written to take the function  $f$  and the number  $c$  from the stack and produce the equation of the tangent line. It might look something like this, which I call TLIN:

```

« 'X' STO DUP 'X' ð 'X' X - * SWAP EVAL 'Y' SWAP - SWAP = 'X'
PURGE
»

```

checksum: # 60926d

For example, to find the line tangent to the curve  $y = x^2 - 4$  at the point (3, 5), type

'X ^ 2 - 4 **ENTER** 3

```

1:      'X^2-4'
3
GN08: IMP0 TLIN ROT0 PCH0 PCH2

```

**TLIN**.

```

1:      'Y-5=6*(X-3)'
GN08: IMP0 TLIN ROT0 PCH0 PCH2

```

This program can be modified to produce the equation of the normal line, just by inserting the commands **INV NEG** after the **ð** symbol, for only the slope is different.

In the graphics environment, the tangent line to a curve at a point can be specified by digitizing the point of tangency, and the line can be drawn on the screen with the curve. The following program, called TANL, does the job.

```

« 1 'N' STO EQ DUP
  IF TYPE 5 ==
  THEN OBJ→ DUP 'N' STO PICK 'CEQ' STO N DROPN
  ELSE 'CEQ' STO EQ 1 →LIST STEQ
  END EQ SWAP C→R DROP DUP 'X' STO 'X' SWAP - CEQ 'X' ð *
  CEQ EVAL + STEQ DRAW 'X' PURGE EQ 1 →LIST + STEQ ( CEQ X )
  PURGE
»

```

checksum: # 9958d

For example, suppose you have just drawn the graph of a function  $f$ , and while the graph is still up you position the cursor at the desired point of tangency (only the x-coordinate of the

point is used by the program). Press **ENTER** to digitize the point, then **ON**, and then press **TANL**. The process can be repeated when the new picture is displayed. This program can also be modified to produce the normal line.

## Function Analysis

The HP 48 has built into it some routines for analyzing a function. After a function  $f$  is plotted, the FCN menu leads to several commands for discovering things about the function. The **ROOT** command finds the zero of the function closest to the position of the cursor; the **SLOPE** command gives the slope of the function's graph at the x-position of the cursor; the **EXTR** command gives the coordinates of the extremum of the function closest to the cursor position. These commands, and others, are described starting on p. 307 of the *Manual*.

### Points of Inflection

A point of inflection of a function  $f$  can usually be spotted from the graph of  $f$ . The following program, INFL, will take a point from the stack and return the nearest point for which  $f''(x) = 0$ . The best way to get the input point is, while viewing the graph, to position the cursor near the inflection point and press **ENTER**. Then press **ATTN** and **INFL** to get the "actual" coordinates of the inflection point.

```
« C→R DROP EQ 'X' PURGE 'X' ÷ 'X' ÷ 'X' 3 ROLL ROOT DUP 'X'  
STO EQ EVAL R→C 'X' PURGE  
»
```

checksum: # 26417d

This program can be followed by TANL, for example, to draw an inflectional tangent to a graph.

### Root Finding

In addition to the methods already mentioned in Chapter 2 for finding the zero of a function, here are two more, the bisection method and Newton's method.

#### Bisection Method

The bisection method is a means of systematically closing in on a zero of a function when an interval containing the zero is known. The method uses the continuity of the function and the fact that the function has opposite signs at the endpoints of the interval to locate the zero. Successively more precise locations are found by repeatedly bisecting the interval and retaining the half that contains the zero.

The following programs enable one to find an arbitrarily short interval containing a zero of a given function  $f$ , once an interval  $[a, b]$  is found for which  $f(a)$  and  $f(b)$  have opposite signs. The first program, FABST is just a simple routine that stores the function and the initial

interval; it has the great value, however, of reminding you how to set up to use the bisection method. The program is

```
« 'B' STO 'A' STO 'F' STO A B 2 →ARRY »
```

checksum: # 22501d

The name of the program reminds you to enter the function  $f$  and the endpoints  $a$  and  $b$  of the interval, in that order, and then press **FABST**. The function and the interval are stored for use by the next program, and the initial interval is placed on the stack for reference.

For example, to set up the bisection method for the function  $f(x) = x - \cos x$  on the interval  $[0, 2]$ , type

```
'X - COS ( X ENTER 0 ENTER 2
2: 'X-COS(X)'
1: 0
2:
FABST
1: [ 0 2 ]
FABST
```

The second program actually does the computation. It finds the midpoint of the interval, tests to see in which half the zero lies, and resets the appropriate endpoint so as to retain the correct half of the original interval. It also checks to see that the initial interval given really does contain a zero, and if we happen to hit the zero exactly, that is also made known. The new interval is placed on the stack. Here is the program, which I call BISCT.

```
« A B + 2 / →NUM 'C' STO A 'X' STO F →NUM B 'X' STO F →NUM *
  IF 0 >
  THEN "SAME SIGN"
  ELSE A 'X' STO F →NUM C 'X' STO F →NUM *
    IF DUP 0 ==
    THEN DROP C "IS A ZERO"
    ELSE
      IF 0 <
      THEN C 'B' STO
      ELSE C 'A' STO
      END A B 2 →ARRY
    END
  END { C X } PURGE
»
```

checksum: # 46422d

For example, once the function and initial interval are stored, press

**BISCT** to get the next interval.

```
2: [ 0 2 ]
1: [ 0 1 ]
FABST
```

Keep pressing **BISCT** for successive intervals.

```

4: [ 0 1 ]
3: [ .5 1 ]
2: [ .5 .75 ]
1: [ .625 .75 ]
FABST BISCT NBISC F H E

```

You can also automate the application of the bisection method a specified number of times. Here is a little program that essentially just presses the **BISCT** button n times for you. It is called NBISC.

```
« 1 SWAP START BISCT NEXT »
```

checksum: # 59739d

This program takes the number n from the stack, and calls BISCT that many times. For example, having stored f, a, and b already, if you want to press **BISCT** eight times the easy way, type

```

8
1: [ 0 2 ]
8
FABST BISCT NBISC F H E

NBISC.
3: [ .71875 .75 ]
2: [ .734375 .75 ]
1: [ .734375 .7421875 ]
FABST BISCT NBISC F H E

```

It is convenient to organize these programs into their own subdirectory, perhaps named BISEC. The complete menu under BISEC might be

```
FABST BISCT NBISC F A B
```

### Newton's Method

Another application of differentiation is Newton's method for estimating a zero of a function. The method starts with the function f and an estimate  $x_0$  of the zero, and uses the iteration formula  $x_{n+1} = \frac{f(x_n)}{f'(x_n)}$  to produce (hopefully) better estimates.

A simple environment for using Newton's method on the HP 48 consists of two programs, FSTO for storing the function f and its derivative, and GUESS for computing  $x_{n+1}$ , given  $x_n$ . Here is FSTO:

```
« XPRG DUP 'F' STO 'X' » 'DF' STO »
```

checksum: # 10189d

Here is GUESS:

```
« 'X' STO X X F EVAL DF EVAL / - 'X' PURGE »
```

checksum: # 34374d

The label FSTO reminds you to enter the expression for  $f(x)$ ; pressing **FSTO** not only stores the function  $f$ , but also computes the derived function  $f'$  and stores it under DF. It first purges any container named X that is accessible, so that the derivative of  $f$  and not its value at some point is stored in DF. Then, knowing that Newton's method requires an initial guess, you enter a number  $x_0$ , close to the zero you wish to find, and press **GUESS**. The new estimate  $x_1$  appears, along with  $x_0$ , so that you can compare them. If you wish another estimate, just press **GUESS** again, and another estimate  $x_2$  appears. Continue pressing **GUESS** until you have the accuracy you need, or until the new estimate does not differ from the previous one.

For example, to find the zero of  $f(x) = x - \cos x$  that is near .7, type

'X - COS ( X <b>ENTER</b>	1: 'X-COS(X)'
	<b>FSTO</b> <b>NEXT</b> <b>NNXT</b> <b>F</b> <b>FPR</b>
<b>FSTO</b> .7	1: .7
	<b>FSTO</b> <b>NEXT</b> <b>NNXT</b> <b>F</b> <b>FPR</b>
<b>GUESS</b> .	2: .7
	1: .739436497848
	<b>FSTO</b> <b>NEXT</b> <b>NNXT</b> <b>F</b> <b>FPR</b>
Press <b>GUESS</b> three or four more times.	4: .739436497848
	3: .739085160465
	2: .739085133215
	1: .739085133215
	<b>FSTO</b> <b>NEXT</b> <b>NNXT</b> <b>F</b> <b>FPR</b>

You can also automate the pressing of **GUESS** a specified number of times. The following program, called NGUESS, takes  $x_0$  from level 2 and a number  $n$  from level 1, and presses **GUESS**  $n$  times for you. It assumes that  $F$  and  $DF$  are already stored.

« 1 SWAP START GUESS NEXT »

checksum: # 56432d

It is convenient to put these programs into their own subdirectory, perhaps called NWTN. The complete menu might be

FSTO    GUESS    NGUESS    F    DF

### Function Tables

One of the first things one learns to do with a function is to create a table of values in order to plot the graph of the function. The following environment makes for easy table creation. If one already has a table of values, the next environment makes for easy analyzing of the table.

### Creating Tables of Function Values

Our first special environment is CRTAB, for "Create Table". This subdirectory contains the following commands and containers:

NVAL      BEGX      STPSZ      FTAB      F      H  
X0          NV

The first four commands are for forming a matrix of evenly-spaced data points for a function.

NVAL stands for "number of values" and takes an integer from the stack and stores it in NV. It tells how many values of x to create. The program is « 'NV' STO ».

BEGX stands for "beginning x-value" and takes a number from the stack and stores it in X0. The program is just « 'X0' STO ».

STPSZ stands for "step size" and takes a number from the stack and stores it in H. The program is « 'H' STO ».

FTAB stands for "function table" and takes a formula for the function from the stack and stores it in F. Then using the other data, it creates NV more values for x, evenly spaced a distance H apart and starting with X0. Then it finds the value of the function at each of those values of x, and presents a matrix with NV + 1 data points in it. FTAB assumes that the formula for the function is written in algebraic form with X as the variable. Here is the program:

```
« 'F' STO X0 'X' STO 0 NV
  START X F EVAL X H + 'X' STO
  NEXT NV 1 + 2 2 →LIST →ARRY 'X' PURGE
»
```

checksum: # 62566d

For example, suppose we want to create a table of values for the function  $f(x) = \sin x$  for the x-values 0.0, 0.1, 0.2, ..., 3.0. Then the beginning x-value is 0, and 30 more values will be created; NV is therefore 30. The step size is 0.1. We create the table by typing:

30 <span style="border: 1px solid black; padding: 2px;">NVAL</span> 0 <span style="border: 1px solid black; padding: 2px;">BEGX</span> 0.1 <span style="border: 1px solid black; padding: 2px;">STPSZ</span> ' <span style="border: 1px solid black; padding: 2px;">SIN</span> X <span style="border: 1px solid black; padding: 2px;">ENTER</span>	<div style="border: 1px solid black; padding: 5px;"> <div style="display: flex; justify-content: space-between; font-size: small;"> <span>RAD</span> <span>HOME</span> <span>NUM</span> <span>TBFNS</span> <span>CRTAB</span> </div> <div style="margin-top: 5px;">             4: 3: 2: 1:           </div> <div style="text-align: right; margin-top: 5px;">'SIN(X)'</div> <div style="display: flex; justify-content: space-between; font-size: x-small;"> <span>NVAL</span> <span>BEGX</span> <span>STPSZ</span> <span>FTAB</span> <span>F</span> <span>H</span> </div> </div>
<span style="border: 1px solid black; padding: 2px;">FTAB</span>	<div style="border: 1px solid black; padding: 5px;"> <div style="display: flex; justify-content: space-between; font-size: small;"> <span>RAD</span> <span>HOME</span> <span>NUM</span> <span>TBFNS</span> <span>CRTAB</span> </div> <div style="margin-top: 5px;">             1: [ [ 0 0 ]           </div> <div style="margin-top: 5px;">             [ .1 9.9833416646...           </div> <div style="margin-top: 5px;">             [ .2 .19866933079...           </div> <div style="margin-top: 5px;">             [ .3 .29552020666...           </div> <div style="display: flex; justify-content: space-between; font-size: x-small;"> <span>NVAL</span> <span>BEGX</span> <span>STPSZ</span> <span>FTAB</span> <span>F</span> <span>H</span> </div> </div>

To read the matrix conveniently, press ▼ to put the matrix in the MatrixWriter.

Difference Tables

A difference table is a table of function values and their differences. The first column of a difference table consists of the values of the variable  $x$ , and the second column consists of the values of the function  $f(x)$ . For sake of simplicity, suppose that the  $x$ -values are evenly spaced a difference  $h$  apart;  $h$  is called the stepsize of the table.

The first difference in a difference table is  $\Delta f(x) = f(x + h) - f(x)$ . The first differences make up the third column in the difference table. The second difference is  $\Delta^2 f(x) = \Delta f(x + h) - \Delta f(x)$ , and occupies the fourth column in the difference table. Higher-order differences are defined similarly. The table does not really have to be evenly spaced in any way, and the following programs allow for that.

A divided difference table is the same as a difference table, except that the first differences are divided by the stepsize, the second differences are divided by the first differences, and so forth. The first divided difference is thus

$$\frac{\Delta f}{h} = \frac{f(x + h) - f(x)}{h}.$$

You can see why such a value might be desirable if you remember the definition of the derivative.

Similarly, a ratio table is a table of function values and their ratios. The first ratio is

$$\rho(x) = \frac{f(x + h)}{f(x)}.$$

Our next special environment is for creating difference, divided difference, and ratio tables of a function given as a matrix of data points. It is called DIFTB for "Difference Tables". This subdirectory contains the following commands and containers:

NDIF      DTAB      DDTAB      RTAB       $\Delta C$

The command NDIF stands for "number of differences" and takes a number from the stack and stores it in  $\Delta C$ . For example, if you type 3 and press NDIF, then the program knows to construct the first, second, and third differences. The program is `« 'ΔC' STO »`.

The command DTAB takes a matrix of data points from the stack and constructs the difference table, making as many new columns as specified by NDIF. DTAB assumes that the matrix has two columns. For example, the output of CRTAB above can be taken directly into DTAB. Here is the program:

```
« DUP SIZE 1 GET → M N
  « M TRN ΔC 2 + N 2 →LIST RDM TRN 'M' STO 1 ΔC
    FOR D 1 N D -
      FOR R 'M(R+1,D+1)-M(R,D+1)' EVAL 'M(R,D+2)' STO
    NEXT
  NEXT M
»
```

»

checksum: # 25379d

The command DDTAB takes a function table from the stack and returns the divided difference table. Here is the program:

```
« DUP SIZE 1 GET → M N
  « M TRN ΔC 2 + N 2 →LIST RDM TRN 'M' STO 1 ΔC
    FOR D 1 N D -
      FOR R 'M(R+1,D+1)-M(R,D+1)' EVAL 'M(R+1,1)-M(R,1)'
    EVAL / 'M(R,D+2)' STO
    NEXT
  NEXT M
```

»

»

checksum: # 13518d

The command RTAB takes a function table from the stack and returns the ratio table. The program is

```
« DUP SIZE 1 GET → M N
  « M TRN 3 N 2 →LIST RDM TRN 'M' STO 1 N 1 -
    FOR R 'M(R+1,2)/M(R,2)' EVAL 'M(R,3)' STO
  NEXT M
```

»

»

checksum: # 64314d





## Chapter 4. Integrals and Their Applications

The HP 48 does a very fine job of evaluating definite integrals. The *Manual* discusses numerical integration starting on p. 432. You can also program your own numerical integration routines, as we will discuss in Chapter 6.

### The Built-in Integrator

Numerical integration requires input of the integrand, the limits, and the variable of integration, and returns the approximate value of the integral and an uncertainty number (which is almost certainly greater than the difference between the given value and the actual value). The desired accuracy is indicated by the FIX mode selected; if the mode is 5 FIX, then a value correct to about 5 decimal places is returned; if the mode setting is STD, the value correct to about 12 significant digits is returned

Suppose we wish to evaluate the definite integral  $\int_a^b f(x) dx$ . Several entry methods are possible, as described below under Antiderivatives. The smallest number of keystrokes is required if the stack is used as follows: put a in level 4, b in level 3, f(x) in level 2, x in level 1, and press  $\int$ . If the HP 48 "knows" the antiderivative of f(x), then the antiderivative is shown evaluated at b and a. To simplify what is shown, press  $\boxed{\text{EVAL}}$  again; the value is given to the accuracy of the mode setting. If the calculator does not happen to "know" the antiderivative of f(x), then the integral is shown in the form  $\int(a,b,f(x),x)$ . If you press  $\boxed{\text{EVAL}}$  at this stage, most likely nothing will happen; press  $\boxed{\rightarrow\text{NUM}}$  to get the approximate value of the integral. The uncertainty number is stored in a container named IERR on the VAR menu.

For example, to evaluate the integral  $\int_0^{1.2} \sin x^2 dx$  with five-decimal-place accuracy, type

<p><math>\boxed{\text{MODES}}</math> 5 <math>\boxed{\text{FIX}}</math> 0 <math>\boxed{\text{ENTER}}</math> 1.2 <math>\boxed{\text{ENTER}}</math></p> <p>' <math>\boxed{\text{SIN}}</math> X <math>\boxed{y^x}</math> 2 <math>\boxed{\text{ENTER}}</math> X</p> <p><math>\int</math></p> <p><math>\boxed{\rightarrow\text{NUM}}</math>.</p> <p>Press <math>\boxed{\text{IERR}}</math> on the VAR menu to see the uncertainty number.</p>	<pre> 3: 0.00000 2: 1.20000 1: 'SIN(X^2)' X STO FIX EQ END SWK BEEP  1: '∫(0,1.20000,SIN(X^ 2),X)' STO FIX EQ END SWK BEEP  1: 0.49612 STO FIX EQ END SWK BEEP  2: 0.49612 1: 4.96019E-6 IERR GRAPH NUMN GEOM CHNG SERE </pre>
---	--

Since the uncertainty number is less than  $5 \times 10^{-6}$ , we do indeed have five decimal places of accuracy.

To avoid having to reset the mode when you are used to working in STD mode in order to evaluate a definite integral in a short time (getting all 11 decimal places of accuracy takes quite a while sometimes), we can write a little program that does it for you. It also resets the mode you were in before you started, and also displays the contents of IERR rather than leaving it on the VAR menu for you to purge later. I call the program INT.D, for "INTEgrate with a certain number of Decimal places".

```
« RCLF → D F
  « D FIX →NUM IERR D 1 + RND 'IERR' DUP PURGE →TAG F STOF
  »
»
```

checksum: # 57739d

This program takes the integral, in the form  $\int(a,b,f(x),x)$ , from level 2 and the number of decimal places desired from level 1 and returns the value of the integral to level 2 and the uncertainty number, labeled IERR, to level 1.

## Areas, Volumes, and Arc Lengths

The value of a definite integral can be interpreted as area, volume, arc length, or something else, depending on how the definite integral is set up. In every case, the calculator can evaluate the definite integral in the same way. It is in setting up the integral that the differences occur, and even there the calculator can be of help. The following examples will give you the idea.

The area between curves  $y = f(x)$  and  $y = g(x)$  over the interval  $[a, b]$  is

$$\int_a^b |f(x) - g(x)| dx.$$

To set up to evaluate this integral numerically, enter the integrand by typing  $f(x)$ , then  $g(x)$ ; then subtract, and then take the absolute value by pressing **ABS** or by typing **A B S** **ENTER**.

For example, the integrand of  $\int_{-\pi}^{\pi/5} |\sin x - \cos x| dx$  is created by typing

```
' S I N ( X ENTER ' C O S ( X ENTER      1: 'ABS(SIN(X)-COS(X))
- A B S ENTER .
```

**GRAPH NUMN GEOM CALC SERIES T.P.M.**

If the function  $f$  is positive over  $[a, b]$  and the region under  $f$  is rotated about the  $x$ -axis to generate a solid of revolution, the volume of the solid is  $\int_a^b \pi[f(x)]^2 dx$ . To enter the integrand, just type in  $f$ , square it, and multiply by  $\pi$ .

For example, to find the volume of the solid generated by revolving the region under  $f(x) = 4 - x^2$  over  $[0, 2]$  about the x-axis, we must evaluate the integral  $\int_0^2 \pi(4 - x^2)^2 dx$ . We can form the integrand by typing

'4 - X ^ 2 [ENTER] [x^2] π \* . 1: 'SQ(4-X^2)\*π'  
GRAPH NUM MODE GEOM CALC RECALL T.M.M

The length of the graph of  $y = f(x)$  over  $[a, b]$  is  $\int_a^b \sqrt{1 + [f'(x)]^2} dx$ . To set up this integrand, enter  $f$ , differentiate it, square it, add 1, and take the square root.

For example, to find the length of the curve  $y = x^2 - 4x + 5$  over  $[-1, 3]$ , we must evaluate the integral  $\int_{-1}^3 \sqrt{1 + [\frac{d}{dx}(x^2 - 4x + 5)]^2} dx$ . The integrand can be created by typing

'X ^ 2 - 4 \* X + 5 [ENTER] 'X [ENTER]  
[D] [x^2] 1 + [√] . 1: '√(SQ(2\*X-4)+1)'  
GRAPH NUM MODE GEOM CALC RECALL T.M.M

A similar formula can be constructed in the case of parametric equations.

## Antiderivatives

The HP 48 has a limited ability to find antiderivatives of functions. Antidifferentiation is done in the context of definite integration, with variable limits. The procedure is to enter the integral and then evaluate it.

There are three ways to enter the integral. The most elegant way is to use the EquationWriter, as illustrated on pp. 429-430 of the *Manual*.

For example, to enter  $\int_a^b \cos(x) dx$  in the EquationWriter, type

[EQUATION] [∫] A [▶] B [▶] [COS] X [▶]  
[▶] X [ENTER] . The integral is translated into algebraic form on the stack.

To evaluate, press [EVAL],

and then [EVAL] again.

1: '∫(A,B,COS(X),X)'  
GRAPH NUM MODE GEOM CALC RECALL T.M.M

1: 'SIN(X)/DX(X)|(X=B)  
-(SIN(X)/DX(X)|(X=A))'  
GRAPH NUM MODE GEOM CALC RECALL T.M.M

1: 'SIN(B)-SIN(A)'  
GRAPH NUM MODE GEOM CALC RECALL T.M.M

A second way to enter the integral is to type it into the command line in algebraic form directly. To enter the above integral again, type

'  $\int$  A , B ,  $\cos$  X  $\blacktriangleright$  , X  $\text{ENTER}$  .      1: '  $\int(A,B,\cos(X),X)$  '  
GRAPH NUMN GEOM1 CHLC SERIE T.W.M

The third way is to put the limits, integrand, and variable of integration on the stack, followed by the  $\int$  command. To evaluate the same integral in this way, type

A  $\text{ENTER}$  B  $\text{ENTER}$  '  $\cos$  X  $\text{ENTER}$  X  
3: 'A'  
2: 'B'  
1: 'COS(X)'  
X  
GRAPH NUMN GEOM1 CHLC SERIE T.W.M

$\int$   
1: 'SIN(X)/dX(X)|(X=B)  
-(SIN(X)/dX(X)|(X=A))'  
GRAPH NUMN GEOM1 CHLC SERIE T.W.M

$\text{EVAL}$  .  
1: 'SIN(B)-SIN(A)'  
GRAPH NUMN GEOM1 CHLC SERIE T.W.M

The HP 48 can give the antiderivatives of polynomials and of any function that is the derivative of a built-in function, as described on p. 429 of the *Manual*. You can experiment to find out what the HP 48 will and won't do in symbolic integration.

## Differential Equations

Many applications of the calculus concern differential equations, in which the derivative of a function is known but the function is not. In case the differential equation can be written in the form

$$\frac{dy}{dx} = m(x, y),$$

then the idea of a "slope field" or "flow" can aid in getting a picture of what the solutions to the differential equation are like.

The differential equation above can be thought of as specifying, at each point (x, y), a slope  $m(x, y)$ . The collection of all slopes so defined is called the *slope field* or *flow* of the differential equation. A picture of the flow can be obtained by selecting a grid of points in the plane and, at each point of the grid, drawing a short line segment having the slope specified by the differential equation.

The environment we create here is the subdirectory FLOW, containing the following commands and containers:

MSTO	ABSTO	CDSTO	SSTO	SLOPE	INTCV
ERACV	FIELD	PLTC	M	S	A
B	C	D	PPAR		

The command MSTO, for "m store", takes the expression m(x, y), written in the two variables X and Y, from the stack and stores it in M.

ABSTO takes two numbers from the stack, the interval in the x-direction, and stores them in A and B. These numbers are used just like XRNG uses them to establish window parameters.

CDSTO takes two numbers from the stack, the interval in the y-direction, and stores them in C and D. It behaves like YRNG does.

SSTO takes a number specifying the step size from the stack and stores it in S. This number sets the grid size.

Here are the programs:

<b>MSTO</b> « 'M' STO »	<b>ABSTO</b> « 'B' STO 'A' STO »	<b>CDSTO</b> « 'D' STO 'C' STO »	<b>SSTO</b> « 'S' STO »
-------------------------------	---	---	-------------------------------

### Slope Fields

The program FLOW sets the plot parameters, establishes the grid, and draws the line segments with the right slopes to show the slope field of the differential equation. It also stores in the container FIELD a picture of the flow. Here is the program FLOW:

```

« A B XRNG C D YRNG ERASE ( # 0d # 0d ) PVIEW DRAX B A -
S / D C - S / → NX NY
« A 'X' STO C 'Y' STO 0 NY
  START 0 NX
  START
    IFERR M EVAL
    THEN MAXR →NUM
    END ATAN → T 'COS(T)+ SIN(T)*i' →NUM S * 2 /
  DUP NEG 'X+Y*i' →NUM + SWAP 'X+Y*i' →NUM + LINE S 'X' STO+
  NEXT A 'X' STO S 'Y' STO+
  NEXT ( X Y ) PURGE CLEAR PICT RCL 'FIELD' STO GRAPH
»
»

```

For example, to create the flow of the equation  $\frac{dy}{dx} = \frac{x+y}{\sin x}$  with the window from -4 to 4 in the x-direction and -3 to 3 in the y-direction and with step size .5, type

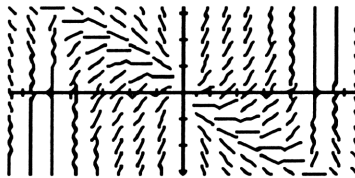
'X+Y [ENTER] ' [SIN] X [ENTER] ÷

[MSTO] 4 [+/-] [SPC] 4 [ABSTO] 3 [+/-]  
[SPC] 3 [CDSTO] .5 [SSTO] [FLOW]

```

RAD                               1USR
{ HOME NU.AN DIFEQ FLOW }
4:
3:
2:
1: '(X+Y)/SIN(X)'
MSTO M1STO 0CSTO 0SSTO FLOW INTCV

```



## Integral Curves

The program INTCV for "Integral Curve" draws an integral curve (solution of the differential equation) on top of the flow diagram, starting with a point specified by the user. The easiest way to specify the point is by use of the cursor while looking at the flow --position the cursor, press [ENTER], then [ATTN], then [INTCV]. This program calls the subroutine PLTC, which uses the Runge-Kutta four-step method to draw the curve.

The program INTCV works well if the solution of the differential equation is a function. If it is a relation, the program may or may not work; you will have to experiment. Just don't automatically believe everything the calculator says!

The command ERACV erases integral curves and restores the slope field to its original form.

Here are the programs:

### INTCV

```

« C→R 'Y0' STO Y0 'Y' STO 'X0' STO X0 'X' STO S 5 / 'H'
STO ( # 0d # 0d ) PVIEW PLTC H NEG 'H' STO X0 'X' STO Y0 'Y'
STO PLTC ( X X0 Y Y0 H K1 K2 K3 K4 ) PURGE GRAPH
»

```

### PLTC

```

« DO X Y R→C M EVAL H * 'K1' STO H 2 / 'X' STO+ K1 2 /
'Y' STO+ M EVAL H * 'K2' STO 'Y' K1 2 / STO- K2 2 / 'Y' STO+
M EVAL H * 'K3' STO H 2 / 'X' STO+ 'Y' K2 2 / STO- K3 'Y'
STO+ M EVAL H * 'K4' STO 'Y' K3 STO- K1 K2 2 * + K3 2 * + K4
+ 6 / 'Y' STO+ X Y R→C LINE
UNTIL A X > B X < OR C Y > OR D Y < OR
END
»

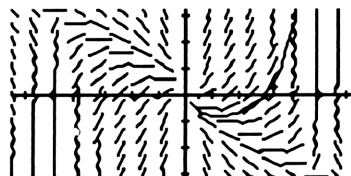
```

## ERACV

« FIELD PICT STO GRAPH »

To draw an integral curve in the previous example passing through the point (.7, -.8), we type

(.7,8



We get an error as the curve approaches the origin.

## Boundary Value Problems

A boundary value problem is a differential equation with an initial or boundary condition and the problem of finding the value of the function for some particular number. We will consider boundary value problems of the following form: Given the differential equation  $\frac{dy}{dx} = f(x, y)$ , whose solution  $y$  is a function of  $x$ , and the condition  $y(a) = \alpha$ , find  $y(b)$ .

The simplest method of estimating a solution to a boundary value problem of this sort is due to Euler. Euler's method is to divide the interval  $[a, b]$  into  $n$  steps, each of length

$$h = \frac{b - a}{n}.$$

At the point  $(a, \alpha)$ , which is on the graph of the solution function  $y$ , we take a step of length  $h$  in the  $x$ -direction along the line with slope  $f(a, \alpha)$ . The new point at which we arrive has  $x$ -coordinate  $x_1 = a + h$  and  $y$ -coordinate  $y_1 = \alpha + hf(a, \alpha)$ . At that point, we evaluate the slope function  $f$  again to find out which direction to go, and take another step in that direction. We thus arrive at a second new point with  $x$ -coordinate  $x_2 = x_1 + h$  and  $y$ -coordinate  $y_2 = y_1 + hf(x_1, y_1)$ . We continue in this way until we have taken  $n$  steps. At each step,  $x_{k+1} = x_k + h$  and  $y_{k+1} = y_k + hf(x_k, y_k)$ .

It is easy to implement Euler's method on the HP 48. We start with a subdirectory that we call EULER whose menu consists of

FABST	NSTO	$\alpha$ STO	EULR	F	N
A	B	$\alpha$	H		

The containers F, N, A, B,  $\alpha$ , and H hold the corresponding numbers or functions.

The program EULR creates a list of the points  $(x_k, y_k)$ , starting with the point  $(a, \alpha)$ . Here is the program:

```
« α 'Y' STO A 'X' STO A α 1 N
  FOR K F EVAL H * H 'X' STO+ 'Y' STO+ X Y
```



NEXT N 1 + 2 2 →LIST →ARRY { X Y } PURGE

»

checksum: # 21682d

The command STO+ is found on the *right-shifted* MEMORY menu.

αSTO is just « α STO ».

NSTO stores N and creates H: « 'N' STO B A - N / 'H' STO ».

FABST is for storing the function f and the numbers a and b:

« 'B' STO 'A' STO 'F' STO »

The function f is written in terms of the variables X and Y.

For example, suppose we are trying to solve the boundary value problem "Find  $y(0.3)$ , given  $\frac{dy}{dx} = 4y^{2/3}$  and  $y(0) = 0.2$ ." If we decided to use  $N = 3$ , we would type

'4\*y^(2/3) [ENTER] 0 [ENTER] .3

```

RAD                               IUSR
{ HOME NU.AN DIFEQ EULER }
3:
2:                               '4*Y^(2/3)'
1:                               0
.3
FABST NSTO αSTO EULR F N

```

[FABST] .2 [αSTO] 3 [NSTO] [EULR]

```

RAD                               IUSR
{ HOME NU.AN DIFEQ EULER }
1: [[ 0 .2 ]
   [ .1 .33679807573...
   [ .2 .53042826011...
   [ .3 .79253474182...
FABST NSTO αSTO EULR F N

```

Thus  $y(0.3)$  is approximately 0.7925. This procedure should be repeated with  $N = 6$  to get a comparative value. You will have to use ▼ to read the last entry in the matrix.

## **Chapter 5. Transcendental Functions**

All of the elementary functions are built into the HP 48, and evaluating a function is just the push of a button. For more complicated settings, like tabular functions, methods of curve fitting can be devised.

### **LN, e, and EXP**

The natural logarithm function is LN. To find the natural logarithm of x, enter x and press **LN**. If x is not a positive number, then LN returns a complex number result.

The number e is the base of the natural logarithm, and satisfies the property  $\ln e = 1$ . The calculator knows e as a symbolic constant; enter a lower-case e, and the calculator puts 'e' on the stack. Press **→NUM** to see the value of e.

The natural exponential function is  $f(x) = e^x = \exp(x)$ . To evaluate  $e^x$ , enter x and press **e<sup>x</sup>**. The key sequence 1 **e<sup>x</sup>** also gives the value of the number e.

Other functions on the MTH HYP menu of the HP 48, described on p. 137 of the *Manual*, are related to LN and EXP.

The derivatives of the LN and EXP functions are built into the calculator, and can be used in differentiation and integration problems just as other functions are.

It may be that a number on the stack is the logarithm or exponential of a rational number. The program **« LN →Q EXP »**, which is called **→EXQ**, will transform the decimal into the exponential of a rational number. The program **« EXP →Q LN »**, which is called **→LNQ**, will transform the decimal into the logarithm of a rational number. Both of these programs give spurious results if the number on the stack is not actually what you thought it was.

### **Inverse Trigonometric Functions**

The inverse trigonometric functions were discussed in Chapter 2, and the derivatives of ASIN, ACOS, and ATAN are known to the calculator. The derivatives of the other inverse trig functions can be computed from these, using the formulas given in Chapter 2.

### **Hyperbolic Functions**

The hyperbolic functions SINH, COSH, and TANH are given on the MTH HYP menu, along with their inverses. The other hyperbolic functions are  $\coth x = \frac{1}{\tanh x}$ ,  $\operatorname{sech} x = \frac{1}{\cosh x}$ , and  $\operatorname{csch} x = \frac{1}{\sinh x}$ . Their values are found as for other functions; just enter x and press the appropriate button, and then use the **1/x** button for the latter three.

## Applications

Many applications of the transcendental functions are based on formulas that lend themselves well to treatment by the Solver. For example, the exponential growth model,

$$Q = Q_0 e^{kt},$$

can be typed as 'Q=Q0\*EXP(K\*T)'. Put this equation on the stack and use the program SV to get it into the Solver. The values of the known variables for a given problem can be stored easily and the remaining variable solved for.

An alternative way to use the Solver on an equation like this is to create a subdirectory into which you can put the formula to be used, together with containers for the variables in the formula. To use the Solver, enter the subdirectory and press **SOLVR**. Then when you get through using the Solver, just exit the subdirectory and leave the variables there. For example, the above equation might be placed in a subdirectory called EXGR (for "EXponential GRowth model"), the complete menu of which might look like

Q            Q0            K            T            EQ

Store the equation 'Q=Q0\*EXP(K\*T)' in EQ, and store zeros in the other containers, just to get them on the menu. Then, whenever you want to work with the exponential growth model, enter the subdirectory EXGR,

```

2:
1:
  Q  Q0  K  T  EQ  QUIT

```

(QUIT is the program « UPDIR ») and then press **SOLVR** on the SOLV menu.

```

2:
1:
  X  Q0  K  T  EQ  EQ

```

The equation is automatically set up for you, because it is already stored in EQ. Similar subdirectories could be set up for other applications, such as the exponential decay model,

$$Q = Q_0 e^{-kt},$$

the logistic model,

$$Q = \frac{LQ_0}{Q_0 + (L - Q_0)e^{-kLt}},$$

and so forth.

## Curve Fitting

Here is a special environment that allows for handling functions in an experimental way or for fitting to data. It was suggested by MS-DOS software called Twiddle, created by Dave Lovelock at the University of Arizona. In the HP 48 version, we allow for creating a function with parameters and graphing it, for plotting the data points of a function given as a table of values, and fitting curves to the data points.

This environment is called TWIDL, and consists of the following commands and containers:

FSTO	ASTO	BSTO	CSTO	draw	FRCL
SAVE	ERASC	STODT	SCTRP	P2LN	LNREG
EXREG	LOGREG	PWREG	POLYREG	SPLCV	POLYFIT
SPLIN	A	B	C	ABC	POINT
SCAT	ΣDAT	EQ	PPAR	ΣPAR	

Some of these names are too long for all their letters to show, and the lower case letters in 'draw' appear as upper case letters on the menu. All of these commands and containers will be discussed in the appropriate sections below, except for PPAR and ΣPAR, which contain parameters you need not be concerned with here.

### Functions with Parameters

The first page of commands is for working with functions having parameters. For example, the function  $f(x) = A \sin(Bx + C)$  is a function of  $x$ , but has the unspecified constants  $A$ ,  $B$ , and  $C$  in its formula. We call  $A$ ,  $B$ , and  $C$  parameters; for any combination of values of the parameters, a specific function  $f$  is obtained. By looking at the function  $f$  for various values of the parameters, we discover properties of the family of sine waves. Similar things can be done with other families of functions.

FSTO is for "F Store", and invites you to enter a function. It takes a formula, written in terms of the variable  $X$  and the parameters  $A$ ,  $B$ , and  $C$ , and stores it in the container  $EQ$ . One or more of the parameters may be missing; a missing parameter simply does not affect the function. The program is « STEQ ABC ».

ASTO, BSTO, and CSTO are for storing values of the parameters. Each of them takes a number from the stack and stores it in the corresponding container, and then displays the current values of all the parameters. One value may be changed while the others remain unchanged. The programs are

```
ASTO:  « 'A' STO ABC »  
BSTO:  « 'B' STO ABC »  
CSTO:  « 'C' STO ABC »
```

draw, which appears as DRAW on the menu, draws the graph of the function, using the current values of the parameters. It draws over whatever is already in PICT, so that several graphs may be viewed at the same time. The program is « DRAX DRAW GRAPH ».

FRCL is for "F Recall", and puts the contents of  $EQ$  on the stack, in case you want to edit it or simply find out what is there. It is simply « EQ ».

SAVE on the second page of TWIDL is for saving the current contents of PICT. It stores the current picture in the container SCAT. The program is « PICT RCL 'SCAT' STO ».

ERASC is for "Erase Curve", and reverts to the last-saved picture. That is, ERASC puts the contents of SCAT back into PICT. If you save a picture, for example, and then draw some more curves on top of it, you can erase the additional curves by using ERASC. The program is « ERASE SCAT PICT STO GRAPH ».

Several of the above programs call the routine ABC, which displays the current values of those containers. (Because ABC freezes the display, the calculator appears to act differently after its use, but don't be alarmed.) Here is the program:

```
« A "A" →TAG 1 DISP B "B" →TAG 2 DISP C "C" →TAG 3 DISP 1
FREEZE 2 FREEZE
»
```

checksum: # 18034d

### Plotting Data Points

The next two commands are for working with data points in a graphical setting. They allow for plotting the points of a data set, so that a visual representation may be made and so that curves may be drawn to fit the data points.

STODT stands for "Store Data", and takes a matrix of data points from the stack and stores it in  $\Sigma$ DAT. If the matrix has more than two columns, only the first two will be used; it will be assumed that the first column contains values of  $x$ , and that the second column contains the corresponding values of  $f(x)$ . The program is just « STO $\Sigma$  ».

SCTRP stands for "Scatterplot", and plots the data points in  $\Sigma$ DAT so that the viewing rectangle just fits the plotted points. Then, so that data points will not disappear if a curve is drawn through them, small circles are drawn around the data points. Here is the program:

```
« SCATRPLOT  $\Sigma$ DAT OBJ→ 1 GET 1 SWAP
  START POINT
  NEXT FUNCTION GRAPH
»
```

checksum: # 4096d

Here is the program POINT, called by SCTRP:

```
« R→C C→PX # 2d 0 'π*2' →NUM ARC »
```

checksum: # 381d

### Fitting Curves to Data

The remaining commands in TWIDL are for fitting a curve to data points. Once data points are plotted, then a function may be guessed that will pass through or close to the data points. If the function involves parameters, they may be adjusted to get a better fit. That is, the commands on the first page of TWIDL may be used to get a curve close to the data points.

The remaining commands create curves of a specific type, relative to data points.

### The Line on Two Points

If two points are given, the command P2LN draws the line through those two points. The usual use of this command is to enter the points from the screen with the cursor. This is done by positioning the cursor on the desired point and pressing **ENTER**. When this is done twice, two points are on the stack. Then press **ATTN** to exit the picture, and then press **P2LN** to get the line through the two points. Here is the program:

```
« C→R 1 3 →ARRY SWAP C→R 1 3 →ARRY CROSS → L
  « L 3 GET L 2 GET / NEG L 1 GET L 2 GET / NEG 'X' * + FSTO
draw
  »
»
```

checksum: # 37686d

### Least Squares Curves

A least squares curve for a set  $\{(x_i, y_i)\}$  of data points is the graph of a function  $y = p(x)$  such that the sum of the squares of the errors,

$$E = \sum_{i=1}^n [y_i - p(x_i)]^2$$

is a minimum. The finding of a least squares curve is called regression analysis. The following table identifies the types of regression curves:

linear regression	$p(x) = a + bx$
exponential regression	$p(x) = ae^{bx}$
logarithmic regression	$p(x) = a + b \ln x$
power regression	$p(x) = ax^b$
polynomial regression	$p(x) = a + bx + cx^2 + \dots + dx^n$

In each case, the data points are used to determine the parameters  $a, b, c, \dots$  that minimize the error  $E$ .

In TWIDL, the built-in regression properties of the HP 48 are used to draw regression curves on a scatterplot. Each of the commands uses the data stored in  $\Sigma$ DAT to create the function  $p(x)$  of the model chosen.

LNREG draws the least squares line (linear function) on the scatterplot. Linear regression is the most commonly used regression method. LNREG takes no argument from the stack, but uses only the data stored in  $\Sigma$ DAT. The program is `« LINFIT  $\Sigma$ LINE FSTO draw »`.

EXREG draws the least squares exponential curve. It uses only the contents of  $\Sigma$ DAT also. Its program is `« EXPFIT  $\Sigma$ LINE FSTO draw »`.

LOGREG draws the least squares logarithmic curve. It also uses only  $\Sigma$ DAT. The program is `« LOGFIT  $\Sigma$ LINE FSTO draw »`.

PWREG draws the least squares power curve, using only  $\Sigma$ DAT. Its program is `« PWRFIT  $\Sigma$ LINE FSTO draw »`.

POLYREG draws the least squares polynomial of the degree specified. It takes the degree from the stack and uses the contents of  $\Sigma$ DAT to get the least squares polynomial. The program is « POLYFIT FSTO draw ». The program POLYFIT that it calls is

```
« → N
  «  $\Sigma$ DAT DUP SIZE 1 GET → D K
    « [[ 0 ]] N 1 + DUP 2 →LIST RDM 'M' STO K 'M(1,1)' STO
   $\Sigma$ Y 1 N
    FOR P 0 'XI' STO 0 'XY' STO 1 K
      FOR I 'D(I,1)' EVAL P ^ DUP 'XI' STO+ 'D(I,2)'
    EVAL * 'XY' STO+
      NEXT XY 1 P 1 +
      FOR J XI 'M(J,P+2-J)' STO
      NEXT
    NEXT N 1 + →ARRY N 1 + N 2 *
    FOR P 0 'XI' STO 1 K
      FOR I 'D(I,1)' EVAL P ^ 'XI' STO+
      NEXT P N - 1 + N 1 +
      FOR J XI 'M(J,P+2-J)' STO
      NEXT
    NEXT M / 'M' STO 'M(N+1)' EVAL N 1
    FOR R 'X' * 'M(R)' EVAL + -1
    STEP { XY XI M } PURGE
  »
»
»
```

checksum: # 32070d

### Cubic Splines

Another method of fitting a curve to data points is by piecing together cubic curves in a smooth way so that they form a curve passing exactly through all the data points. Such a curve is called a cubic spline. The command SPLCV creates and draws the cubic spline for a set of data points. It takes no argument from the stack, using only what is stored in  $\Sigma$ DAT. Here is the program:

```
« RCL $\Sigma$  SPLIN DUP SIZE 1 GET → M N
  « 1 N 1 -
    FOR I 'X' 'M(I,1)' EVAL - 'Y' STO 'M(I,5)' EVAL Y *
    'M(I,4)' EVAL + Y * 'M(I,3)' EVAL + Y * 'M(I,2)' EVAL + 'Y'
  STO
  «
    IF X 'M(I,1)' EVAL > X 'M(I+1,1)' EVAL < AND
    THEN Y
```

```

        ELSE (0,0)
        END
    » STEQ DRAW
NEXT 'Y' PURGE GRAPH
»
»

```

checksum: #22706d

The program SPLIN that is called by SPLCV is

```

« DUP SIZE 1 GET → M N
  « M TRN ( 5 N ) RDM TRN 'M' STO 1 N 1 -
    FOR I 'M(I+1,1)-M(I,1)' EVAL
    NEXT N 1 - →ARRY 'H' STO 1 N 2 -
      FOR I 'M(I+2,2)*H(I)-M(I+1,2)*(M(I+2,1)-
M(I,1))+M(I,2)*H(I+1)' EVAL 3 * 'H(I)*H(I+1)' EVAL /
      NEXT N 2 - →ARRY 'A' STO 1 N 3 *
      START 0
      NEXT ( N 3 ) →ARRY 'L' STO 1 'L(1,1)' STO 1 'L(N,1)'
STO 2 N 1 -
      FOR I '2*(M(I+1,1)-M(I-1,1))-H(I-1)*L(I-1,2)' EVAL
      'L(I,1)' STO 'H(I)/L(I,1)' EVAL 'L(I,2)' STO '(A(I-1)-H(I-
1)*L(I-1,3))/L(I,1)' EVAL 'L(I,3)' STO
      NEXT N 1 - 1
      FOR I 'L(I,3)-L(I,2)*M(I+1,4)' EVAL 'M(I,4)' STO
      '(M(I+1,2)M(I,2))/H(I)' EVAL 'H(I)*(M(I+1,4)+2*M(I,4))' EVAL
      3 / - 'M(I,3)' STO '(M(I+1,4)-M(I,4))/3/H(I)' EVAL 'M(I,5)'
STO -1
      STEP ( H A L ) PURGE M
  »
»

```

checksum: # 47144d

The remaining containers are A, B, and C, for holding values of the parameters, EQ, for storing the function f, and ΣDAT, for holding the matrix of data points.

We will demonstrate some of the capabilities of TWIDL. Suppose we have the following table of function values:

x =	1	2	3	4	5
f(x) =	1	2	2.5	2	3

We wish to view these data points graphically and draw the least squares line and the least squares parabola through them.

We begin by creating a matrix of the data points:



**MATRIX** 1 **SPC** 1 **ENTER** **▼** 2 **SPC** 2 **RAD** **HOME** **1USR**  
**SPC** 3 **SPC** 2.5 **SPC** 4 **SPC** 2 **SPC** 1:  
 5 **SPC** 3 **ENTER** **ENTER**

1	1	1
2	2	2
3	2.5	2.5
4	2	2

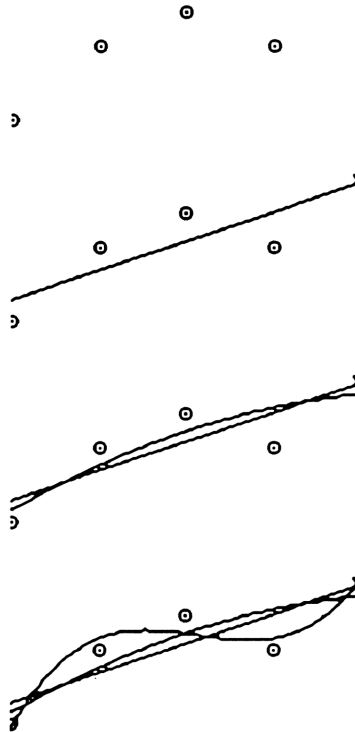
**CLC** **LNRL** **NUM** **NTH** **TWIDL** **DEMO**

Then enter TWIDL, find the second page of the menu, and press **STODT** **SCTRP**

Then press **ON** and **LNREG**

and then **ON** 2 **POLYREG**

This makes us wonder what kind of fit a cubic least squares curve would give, so we type **ON** 3 **POLYREG**



## Chapter 6. Numerical Integration Theory

In first-year calculus, you learn to approximate definite integrals using the trapezoidal rule or Simpson's rule, which are simple approximation techniques that work very well for most definite integrals. Here we will present a few simple programs that develop these rules, as well as the rectangle rules (Riemann sum rules).

Begin by creating a subdirectory for numerical integration. We will call it N.INT for Numerical INTe gration. Type

'N.INT' **ENTER**

1: 'N. INT'  
GRAPH NUMN GEOM CHLO SERIE TMM

CRDIR **ENTER**.

1: N.INT GRAPH NUMN GEOM CHLO SERIE

Then press **N.INT** to enter the new subdirectory.

1: \_\_\_\_\_

Our aim is to approximate the integral  $\int_a^b f(x) dx$ . To do so, we will have to give the calculator the function  $f$  and the limits  $a$  and  $b$ . We will create a little program that makes this easy to do. Type in the following program:

« 'B' STO 'A' STO 'F' STO »

checksum: # 59581d

We will store this program under the label FABST, reminding us to put the entries on the stack in the order  $f$ , then  $a$ , then  $b$ . Once you have entered the above program, typing 'FABST' **STO** will store it for us.

To test our little storage program, let's prepare to evaluate  $\int_1^2 \frac{1}{x} dx$ . Type the following:

'1 ÷ X' **ENTER** 1 **ENTER** 2

2: '1/X'  
1: 1  
2: \_\_\_\_\_  
FABST QUIT

**FABST**.

1: F A B FABST QUIT

You will see three new containers in the menu, F, A, and B. You can press them to recall their contents if you wish to see whether everything worked properly.

## Riemann Sum Rules

The first approximation we want to consider is a Riemann sum rule. This just uses the definition of the definite integral, which is the limit of Riemann sums, to get an approximation as follows. First, the interval  $[a, b]$  is subdivided into a number  $n$  of subintervals, all of the same length

$$h = \frac{b - a}{n}.$$

The subdivision points are then given by

$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_k = a + kh, \dots, x_n = b.$$

Next, in each subinterval a value  $t_k$  is chosen, and a rectangle of height  $f(t_k)$  and width  $h$  is created. The area of the rectangle is approximately the area under the function  $f$ . Finally, the sum of all such areas of rectangles is formed, and that is approximately the value of the integral.

The next step, then, is to tell the calculator now many subintervals we want. The following program stores the number  $n$ , and also calculates the number  $h$  and stores it, using the previously stored values of  $a$  and  $b$ .

```
« 'N' STO B A - N / →NUM 'H' STO »
```

checksum: # 29449d

We will store this program under the label NSTO, reminding us that it stores  $n$ . To test it out, choose the number 4 for  $n$ , typing

4 NSTO.

```
1:
H N NSTO F H E
```

You should see new containers for  $N$  and  $H$ .

The most commonly used Riemann sum rules are the left endpoint rule,

$$\text{LER} = [f(x_0) + f(x_1) + \dots + f(x_{n-1})]h,$$

the right endpoint rule,

$$\text{RER} = [f(x_1) + f(x_2) + \dots + f(x_n)]h,$$

and the midpoint rule,

$$\text{MPR} = [f(\frac{x_0 + x_1}{2}) + f(\frac{x_1 + x_2}{2}) + \dots + f(\frac{x_{n-1} + x_n}{2})]h.$$

In each of these, the numbers  $t_k$  from the subintervals are always spaced a distance  $h$  apart. The next program forms the basis of each of these Riemann sum rules, and creates the sum that each of them uses. We will call this program SUM:

```
« 'X' STO 0 1 N
  START F →NUM + H 'X' STO+
```

```

NEXT H * 'X' PURGE
»

```

checksum: # 3683d

The command STO+ is found on the *right-shifted* MEMORY menu on the HP 48.

This program takes a number from the stack as the starting point, stores it as X, and uses it to evaluate the function f. Then the next value, a distance h to the right, is created, f is evaluated there, and added to the previous value, and so forth. In this way a sum of functional values is created. Finally, the sum is multiplied by h, giving the sum of the areas of the rectangles, and we are done. X is purged to keep the menu clean.

Now we can create the Riemann sum rules very easily. The left endpoint rule starts with the number a, so the following program is all that is needed:

```

« A SUM »

```

We will store this program under the label LER, for Left Endpoint Rule.

The right endpoint rule starts with the point a + h, so the following program suffices:

```

« A H + SUM »

```

We will store this program under the label RER.

All things should be ready for us to test these programs out now. Pressing

```

LER and RER

```

2:	.759523809525
1:	.634523809525
ERR	LER
SUM	H
N	N TO

will give us two estimates for  $\int_1^2 \frac{1}{x} dx$ , using  $n = 4$  subintervals.

## Error Analysis

We know the actual value of this integral, namely  $\ln 2$ . Let's do a little error analysis. To make it easy, we create a little program that immediately shows us how much error there is in our estimates. Store the program « 2 LN - » under the label ERR. Then pressing

```

LER ERR

```

3:	.759523809525
2:	.634523809525
1:	.066376628965
ERR	LER
SUM	H
N	N TO

gives us the error in the left endpoint rule using the current value of n.

First of all, let's see what effect changing n has on the error. With  $n = 4$ , press

```

LER ERR and then RER ERR.

```

2:	.066376628965
1:	-.058623371035
ERR	LER
SUM	H
N	N TO

Then we will multiply  $n$  by some factor, say 5. That makes  $n = 20$ , so type

20     .

4:	.066376628965
3:	-.058623371035
2:	.01265620123
1:	-.01234379877
ERR	LER
SUM	N

Compare these errors with the previous errors. How are they related? Now let's multiply  $n$  by the factor 5 again, making  $n = 100$ . Type

100     .

4:	.01265620123
3:	-.01234379877
2:	.002506249917
1:	-.002493750083
ERR	LER
SUM	N

Compare again. What do you think?

You should have concluded that multiplying  $n$  by 5 divides the error by 5, approximately. In terms of accuracy, if a given  $n$  produces one decimal place of accuracy, it will take an  $n$  10 times as large to produce two decimal places of accuracy.

This brings up the question of whether we can get whatever accuracy we like by increasing  $n$ . Let's investigate. With  $n = 4$ , LER and RER both gave us one decimal place of accuracy, if we rounded off. Try these again, and estimate the time it takes for the calculator to run the programs. I get about a second. So let's suppose that we can get one decimal place of accuracy in one second. That means it will take 10 seconds to get two decimal places of accuracy, 100 seconds to get three, and so forth. What if we want 12 decimal places of accuracy? That requires  $10^{12}$  seconds. Use your calculator to turn that into years. Discouraging, isn't it?

What this little exercise demonstrates is that something so simple as the Riemann sum rules will never give us very much accuracy. We have to be smarter than that.

The integrand  $f(x) = \frac{1}{x}$  has a fairly flat graph, meaning that rectangles do a fair job of approximating the function. If the integrand had a steeper graph, it is clear that rectangles would do a poorer job yet. Thus the error in the Riemann sum rules also depends on the steepness of the graph, which is given by the derivative of the function. Textbooks give the following error estimate: If  $|f'(x)| \leq B$  on  $[a, b]$ , then the error in approximating

$\int_a^b f(x) dx$  by a Riemann sum rule is no greater than  $\frac{B(b-a)^2}{2n}$ .

## Trapezoidal Rule

You probably noticed when computing errors above that the LER and RER programs had about the same amount of error, but in opposite directions. That is, one was an overestimate, and the other, an underestimate. This makes us wonder what would happen if we averaged the two estimates; will the error cancel out?

Let's try it. Store the program

« LER RER + 2 / »

under the label TRAP. This is the trapezoidal rule. Geometrically, the rectangles on the subintervals have been replaced by trapezoids. Try TRAP a few times, and try to determine how multiplying  $n$  by a factor affects the error.

You should discover that multiplying  $n$  by a factor of  $k$  divides the error by a factor of about  $k^2$ . Thus, if a given value of  $n$  gives one decimal place of accuracy, then about 3 times  $n$  will give two decimal places of accuracy. (Here,  $3 \approx \sqrt{10}$ .)

It is also evident that the concavity of the integrand affects the accuracy of the trapezoidal rule, and that is why the second derivative figures into the error formula:

If  $|f''(x)| \leq B$  on  $[a, b]$ , then the error in approximating  $\int_a^b f(x) dx$  by the trapezoidal rule is no greater than  $\frac{B(b-a)^3}{12n^2}$ .

## Midpoint Rule

Another obvious way to improve on the left and right endpoint rules is to use the midpoint of each interval rather than an endpoint. This gives the midpoint rule, whose program is

« H 2 / A + SUM »

Store this under the label MPR. Experiment with MPR a few times, to see how multiplying  $n$  by a factor affects the error. You should discover the same "quadratic" law that was the case with the trapezoidal rule.

## Simpson's Rule

Since TRAP and MRR have similar error behavior, we wonder if combining them could reduce the error still further. When we use both MPR and TRAP, we notice that the error of TRAP is about twice that of MPR, and in the opposite direction. This suggests a weighted average, as given by the program

« MPR 2 \* TRAP + 3 / »

Store this under the label SIMP. That's right; it is Simpson's rule. Use SIMP a few times, to see how the error behaves when  $n$  is multiplied by a factor.

You should discover that multiplying  $n$  by  $k$  divides the error by about  $k^4$ . One reason for this is that, by using the midpoints of the subintervals also, we have essentially doubled the number of subintervals. The textbook error formula for the above program is as follows. If  $|f^{(4)}(x)| \leq B$  on  $[a, b]$ , then the error in approximating  $\int_a^b f(x) dx$  by Simpson's sum rule is

no greater than  $\frac{B(b-a)^5}{2880n^4}$ . The influence of the fourth derivative instead of the third is a surprise.

The complete menu under N.INT is now

SIMP	MPR	TRAP	ERR	RER	LER
H	N	NSTO	F	A	B
FABST					

It might be more convenient to reorder this menu, so that FABST and NSTO are nearer to the front. Purge the program ERR and order the menu as follows:

FABST	NSTO	SIMP	TRAP	MPR	F
RER	LER	H	N	A	B

This is done by creating the list

{ FABST NSTO SIMP TRAP  
MPR F } ENTER

```

RAD          1USR
{ HOME CALC INTE N.INT }
3:
2:
1: { FABST NSTO SIMP
   TRAP MPR F }
SIMP MPR TRAP RER LER H

```

and then typing O R D E R ENTER.

```

RAD          1USR
{ HOME CALC INTE N.INT }
4:
3:
2:
1:
FABST NSTO SIMP TRAP MPR F

```

You may also find ORDER on the MEMORY menu.

## Chapter 7. Sequences and Series

Some aspects of the subjects of sequences and series lend themselves to computation. A few ways in which the HP 48 is useful are presented here.

### Terms of a Sequence

If the terms of a sequence are defined by a formula, then the formula can be evaluated just as a function can be, using the Solver, as discussed in Chapter 2. Thus, as many specified terms of the sequence can be found as desired, just by evaluating.

#### Sequences by Formula

A simple program can be written to create specified terms of a sequence, as well. If the formula, the number of the first term desired, and the number of the last term are supplied, the following program, called SEQ, will create a list of the terms specified.

```
« → A N1 N2
  « N1 N2
    FOR K K 'N' STO A EVAL
    NEXT N2 N1 - 1 + →LIST 'N' PURGE
  »
»
```

checksum: # 30925d

For example, to create the first five terms of the sequence  $\left\{ \frac{n^2}{2n+1} \right\}$ , type

```
'N^2/(2*N+1) [ENTER] 1 [ENTER] 5
2:      'N^2/(2*N+1)'
1:      1
5
TWOC SEQ

1: ( .333333333333 .8
   1.28571428571
   1.77777777778
   2.27272727273 )
TWOC SEQ
```

If you want the terms of the sequence in fraction form, insert the command  $\rightarrow Q$  or  $D \rightarrow Q$  into the program SEQ just before the command NEXT.



## Recursive Sequences

If a sequence is defined recursively, then a program can be written to produce the next term, given the preceding terms on which it depends.

For example, the Fibonacci sequence is defined by the recursion  $a_1 = 0$ ,  $a_2 = 1$ , and  $a_{n+2} = a_n + a_{n+1}$  for any  $n$ . The following program, called FIB, will produce the next term, given any two consecutive terms of the Fibonacci sequence.

« DUP2 + »

For example, with 0 in level 2 and 1 in level 1, pressing **FIB** will produce the next term, 1, in level 1, moving the others up. Repeated applications of FIB produce successive terms of the sequence.

To automate the application of FIB, the following program, called NFIB, will take two terms and the number  $n$  of new terms desired from the stack and return the next  $n$  terms to the stack:

« 1 SWAP START FIB NEXT »

For example, to get the first twelve terms of the Fibonacci sequence, type

0 **ENTER** 1 **ENTER** 10

2:	0
1:	1
10	
<b>FIB NFIB FIB TWRD SED</b>	

**NFIB**.

4:	21
3:	34
2:	55
1:	89
<b>FIB NFIB FIB TWRD SED</b>	

If you want a list of the next  $n$  terms of the sequence, the following program, FIBL, will give it to you.

« → N « N NFIB N 2 + →LIST » »

Type

0 **ENTER** 1 **ENTER** 10

2:	0
1:	1
10	
<b>FIB NFIB FIB TWRD SED</b>	

**FIBL**.

1:	{ 0 1 1 2 3 5 8 13
	21 34 55 89 }
<b>FIB NFIB FIB TWRD SED</b>	

The above techniques used to produce the Fibonacci sequence will work with any recursively-defined sequence. In fact, the programs NBISC and NGUESS, for approximating zeros of a function by the bisection method or by Newton's method, respectively, are just programs for producing some more terms of a sequence. You must

determine the exact nature of the programs to use each time, but these examples make the process evident.

## Partial Sums

The  $n^{\text{th}}$  partial sum of a series is just the sum of the first  $n$  terms of the series. The HP 48 has a built-in command,  $\Sigma$ , for computing a partial sum. This is described and illustrated on pp. 423-426 of the *Manual*.

For example, to compute the sum  $\sum_{n=1}^{100} \frac{1}{n^2}$ , we just enter the sum and evaluate it. The

*Manual* illustrates using the EquationWriter environment, so we will illustrate using the stack. Type

'N [ENTER] 1 [ENTER] 100 [ENTER]  
'1 / N ^ 2 [ENTER]

$\Sigma$ .

```

4: 'N'
3: 1
2: 100
1: '1/N^2'
FIEL NPIE FIE TIME SEC
2:
1: 1.63498390017
FIEL NPIE FIE TIME SEC

```

## Geometric Series

A geometric series is a series of the form  $\sum_{n=0}^{\infty} ar^n$ . If  $|r| < 1$ , then the series converges, and has sum  $\frac{a}{1-r}$ . Thus, for geometric series, we can find not only terms and partial sums, as above, but also the sum. The environment of the Solver, with the formula  $\frac{a}{1-r}$ , provides an excellent setting for finding the sums of various geometric series.

## Taylor Polynomials

The HP 48 has built into it the means for computing Taylor polynomials of functions. The command TAYLR on the ALGEBRA menu takes a function, the independent variable, and the degree from the stack, and returns the Taylor polynomial of the function of the specified degree in the independent variable, centered at zero.

For example, to compute the Taylor polynomial of  $f(x) = \cos \frac{5x}{2}$  of degree 4 about 0, type

```

' C O S ( 5 * X / 2 [ENTER] ' X [ENTER] 4
2:      'COS(5*X/2)'
1:      'X'
4
COLCT EXPN ISOL DUWG SHOW TAYLR
TAYLR.
1: '1-3.125*X^2+
   39.0625/4!*X^4'
COLCT EXPN ISOL DUWG SHOW TAYLR

```

You may find it interesting to apply the command  $\rightarrow Q$  or COLCT  $\rightarrow Q$  afterward.

The Taylor polynomials for the functions  $e^x$ ,  $\sin x$ ,  $\cos x$ , and  $\ln(1 + x)$  are used quite often. You may find it convenient to store those polynomials in general form for use in the Solver.

For example, for  $e^x$  enter the Equation Writer and type  $\Sigma K = 0 \rightarrow N \rightarrow X^K / K!$

[ENTER] and store the sum for later use. When you put the sum into the Solver, you may specify the values of  $N$  and  $X$ , and the Solver will compute the sum for you very quickly. (Do not specify a value for  $K$ —it is a dummy index.) This is particularly convenient for those problems that ask you to tell how many terms of the Taylor polynomial are needed to achieve a certain accuracy; put in consecutive values for  $N$  and watch the output.

Since the Taylor polynomials computed by the calculator are centered at zero, they are partial sums of Maclaurin series. Thus the TAYLR command can be interpreted as creating the specified number of terms of the Maclaurin series of the function given.

If you want to compute a Taylor polynomial about some other point  $c$ , the substitution of  $y + c$  for  $x$  before the computation and  $x - c$  for  $y$  afterward will do it. For example, to find

the Taylor polynomial of  $f(x) = \cos \frac{5x}{2}$  of degree 4 about 1, type ' C O S ( 5 \* X / 2

[ENTER] ' Y + 1 [ENTER] ' X [STO] [EVAL] ' Y [ENTER] 4 [TAYLR] ' X [PURGE] ' X - 1 [ENTER] ' Y [STO] [EVAL] ' Y [PURGE] .

This can be automated, of course; consider the following program, called TAYC.

```

« → N C
  « 'Y' C + 'X' STO EVAL 'Y' N TAYLR 'X' PURGE 'X' C - 'Y'
STO EVAL 'Y' PURGE
  »
»

```

checksum: # 35391d

This program takes the function  $f$ , written in terms of the variable  $X$ , from level 3, the degree  $n$  from level 2, and the center  $c$  from level 1, and returns the Taylor polynomial of degree  $n$  for  $f$  centered at  $c$ .

For example, to redo the above example, and get the Taylor polynomial of  $f(x) = \cos \frac{5x}{2}$  of degree 4 about  $c = 1$ , type

```

' C O S ( 5 * X / 2 [ENTER] 4 [ENTER] 1
2: 'COS(5*X/2)'
1: 4
1
TAYC

1: '-.801143615547-
1.49618036026*(X-1)
+2.50357379859*(X-1)
)^2+1.5585212086*(X
TAYC

```

If the function given involves other variables than the one specified for use with TAYLR, the polynomial's coefficients are given in terms of those variables. For use with TAYC, the function given can involve other variables except for Y, and must use X as the variable supplied to TAYLR.



## Chapter 8. Conic Sections and Polar Coordinates

The HP 48 can produce graphs of conic sections, parametric equations, and polar coordinates. It can also be programmed to produce the graph of any implicitly-defined function.

### Graphs of Conics

The conic sections, with equations of the form

$$ax^2 + bxy + cy^2 + dx + ey + f = 0,$$

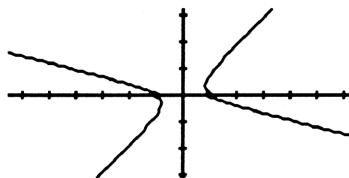
are special cases of implicitly-defined functions, with equations of the form

$$f(x, y) = 0.$$

The HP 48 has built-in an environment for plotting the graphs of conic sections, which it calls the CONIC type. Type in the function  $f(x, y)$  and store it as the equation by pressing **[STEQ]** on the PLOT menu. Then specify CONIC type by pressing **[CONIC]** on the PLOT PTYPE menu. Then draw the graph by pressing **[DRAW]** on the PLOT PLOT menu. (ERASE the previous graph first if you wish.)

For example, to get the graph of  $x^2 + 3xy - 4y^2 = 1$ , type

```
'X^2+3*X*Y-4*Y^2-1' [ENTER] 1: 'X^2+3*X*Y-4*Y^2-1'
                                TOPWR T.W.M GRAPH NUM.MN GEOM1 CHLC
[PLOT] [PTYPE] [CONIC] [STEQ] 2:
                                1:
                                PLOT PTYPE NEW EDCX ETER CWT
                                -----
                                Indep: 'X'
                                Depnd: 'Y'
                                x:      -6.5      6.5
                                y:      -3.1      3.2
                                ERASE GRAPH AUTO MARK MARKS INDEP
[PLOT] [ERASE]
                                -----
[PLOT] [DRAW]
                                -----
```



### Implicit Functions

To plot the graphs of implicitly-defined functions that are not conic sections, a replacement for the DRAW command is needed. The following program, IMPG, is a successful

substitute for DRAW. The plot parameters, etc., are set from the PLOT PLOT menu, but then go to the VAR menu and use IMPG instead of DRAW.

```

« DRAX PPAR OBJ→ 6 DROPN C→R 'Y2' STO 'X2' STO C→R 'Y1' STO
'X1' STO X1 Y2 R→C PVIEW X2 X1 - 5 * 131 / 'DX' STO Y2 Y1 -
5 * 62 / 'DY' STO Y1 'Y' STO 1 12
  START X1 'X' STO EQ →NUM 'Z1' STO DY 'Y' STO+ EQ →NUM 'Z2'
STO 1 26
  START DX 'X' STO+ EQ →NUM 'Z4' STO Y DY - 'Y' STO EQ
→NUM 'Z3' STO
  IF Z1 Z2 * 0 < Z1 Z3 * 0 < OR Z1 Z4 * 0 < OR
  THEN 1 5
    FOR K X DX - DX 5 / K * + 'T' STO Z3 Z1 - K * 5
/ Z1 + 'T1' STO Z4 Z2 - K * 5 / Z2 + 'T2' STO
    IF T1 T2 * 0 <
    THEN T DY T1 * T1 T2 - / Y + R→C PIXON
    END
  NEXT 1 5
  FOR J Y DY 5 / J * + 'T' STO Z2 Z1 - J * 5 / Z1 +
'T1' STO Z4 Z3 - J * 5 / Z3 + 'T2' STO
  IF T1 T2 * 0 <
  THEN X DX - DX T1 * T1 T2 - / + T R→C PIXON
  END
  NEXT
  END Z4 'Z2' STO Z3 'Z1' STO DY 'Y' STO+
  NEXT
  NEXT ( X1 X2 Y1 Y2 DX DY Z1 Z2 Z3 Z4 T T1 T2 X Y ) PURGE
GRAPH
»

```

checksum:# 64113d

The above program is based on an interpolation routine that tests for sign changes in small boxes, skipping the interiors of the boxes that have no sign changes in order to save time. Where sign changes occur, every pixel is tested for the sign change, and an unbroken graph is created.

## Rotation of the Coordinate System

In the equation

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

of a conic section, the  $xy$ -term can be eliminated by a rotation of the coordinate system. The angle through which the coordinate system should be rotated is given by

$$\cot 2\alpha = \frac{a - c}{b},$$

and the rotation equations

$$\begin{aligned} x &= u \cos \alpha - v \sin \alpha \\ y &= u \sin \alpha + v \cos \alpha, \end{aligned}$$

when substituted for  $x$  and  $y$  in the original equation, give the new equation of the conic section in the  $uv$ -coordinate system. The process of carrying out these operations can be very tedious. Using the symbol-manipulation power of the HP 48, this process can be reduced to a program.

The following program, called ROTCO, for "ROTate CONic", takes the equation of a conic from level 4 and the coefficients  $a$ ,  $b$ , and  $c$  from levels 3, 2, and 1, and returns the "simplified" equation in the  $uv$ -coordinate system. This program also uses the program EXCO, given in the *Manual*, starting on p. 569.

```
« ROT SWAP - SWAP / DUP 0
  IF ==
    THEN DROP  $\pi$  4 /  $\rightarrow$ NUM
    ELSE INV ATAN 2 /
    END  $\rightarrow$  T
    « 'U' T COS * 'V' T SIN * - 'X' STO 'U' T SIN * 'V' T COS
    * + 'Y' STO EVAL EXCO
    » { X Y } PURGE
  »
```

checksum: # 43082d

The equation of the conic should be entered in terms of  $X$  and  $Y$ , and the coefficients  $a$ ,  $b$ , and  $c$  must be entered in order. If  $b = 0$ , the rotation is unnecessary, and the attempt to use the program will result in a "division by zero" error. If a  $uv$ -term appears in the result, it is because of round-off error, and it can be eliminated.

For example, to eliminate the  $xy$ -term in the equation  $x^2 + 5xy - y^2 + x = 8$ , we type

```
' X ^ 2 + 5 * X * Y - Y ^ 2 + X = 8 [ENTER]
1 [ENTER] 5 [ENTER] 1 [CHS]
[ROTCO].
```

3: 'X^2+5\*X\*Y-Y^2+X=8'  
 2: 1  
 1: 5  
 -1  
 [ONCE] [IMPO] [TLIN] [ROTCO] [PCHS] [PCH2]  
 1: '2.69258240357\*U^2+  
 .0000000000006\*U\*V-  
 2.69258240357\*V^2+  
 .82806723047\*U-  
 [ONCE] [IMPO] [TLIN] [ROTCO] [PCHS] [PCH2]

The program takes a few moments (it's the EXCO part), so be patient.



## Parametric Equations

A pair of parametric equations

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases}, t \in [a, b],$$

define a curve in the plane from  $(f(a), g(a))$  to  $(f(b), g(b))$ , provided that  $f$  and  $g$  are continuous functions. The HP 48 can help us find the slope of such a curve, and sketch its graph.

### Chain Rule

The parametric form of the chain rule is

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{g'(t)}{f'(t)}$$

and gives the slope of the parametric curve. The following little program for the HP takes the functions  $f$  and  $g$  from the stack and returns the derivative  $dy/dx$ . The name of this program is PCHR, for "Parametric CHain Rule."

```
« 'T' ð SWAP 'T' ð / »
```

checksum: # 44999d

The functions for  $x$  and  $y$  should be entered in terms of  $T$ , and in the order  $x$ , then  $y$ .

For example, if the parametric equations are

$$\begin{cases} x = \sin t \\ y = t^3 \end{cases},$$

then to find  $dy/dx$  we type

```
'S I N ( T [ENTER] ' T ^ 3 [ENTER]
```

```
2:      'SIN(T)'
1:      'T^3'
[ON] [IMP] [TLIN] [DOT] [PCHR] [PCR2]
```

```
[PCHR].
```

```
1:      '3*T^2/COS(T)'
[ON] [IMP] [TLIN] [DOT] [PCHR] [PCR2]
```

We can even write a program to find the second derivative in parametric equations. If we let  $y' = \frac{dy}{dx}$ , then  $\frac{d^2y}{dx^2} = \frac{dy'/dt}{dx/dt}$ . The following program, called PCR2 (for "Parametric Chair Rule, 2nd derivative"), will then do the trick. Just like PCHR, it starts with the parametric functions  $f$  and  $g$  on the stack.

```
« SWAP DUP → F « SWAP PCHR F SWAP PCHR » »
```

checksum: # 11824d

You may want to use EXCO to simplify the result.

## Graphs

The HP 48 has built-in a procedure for drawing the graph of a pair of parametric equations, as described on p. 332 of the *Manual*. You must specify PARAMETRIC plot type by pressing **PARA** on the PLOT PTYPE menu. Then write the functions  $x = f(t)$  and  $y = g(t)$  in the form  $f(t) + i g(t)$ , a complex number with real part  $f(t)$  and imaginary part  $g(t)$ . Store  $f(t) + i g(t)$  as the equation by pressing **STEQ** on the PLOT menu. Finally, specify the interval over which  $t$  is to vary; if  $t$  runs from  $a$  to  $b$ , type the list  $\{ T \ a \ b \}$  and press **INDEP** on the PLOT PLOTR menu. Then press **DRAW**.

For example, to get the graph of the pair of equations  $x = 3 \cos t$ ,  $y = \sin t$ , with  $t$  running from 0 to  $2\pi$ , type

**PLOT** **PTYPE** **PARA** ' 3 \* **COS** T  
**►** + i \* **SIN** T **ENTER**

**STEQ** **PLOTR** { T 0 6.29 **ENTER**

**INDEP**

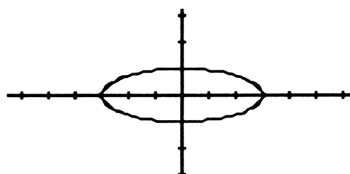
**ERASE** **DRAW**.

1: '3\*COS(T)+i\*SIN(T)'  
**PLOT: PTYPE NEW EQN: STEQ CMY**

1: { T 0 6.29 }  
**ERASE DRAW AUTO WAND WAND INDEP**

Indep: { T 0 6.29 }  
x: -6.5 6.5  
y: -3.1 3.2

**ERASE DRAW AUTO WAND WAND INDEP**



## Polar Coordinates

Routines for handling polar coordinates and transforming to rectangular coordinates and back again are built into the HP 48. This makes both computation and graphing easy to do.

The transformation from rectangular to polar coordinates and back again was discussed in Chapter 2 when we dealt with vectors. Further discussion is found on pp. 169ff of the *Manual*.

## Graphs

The HP 48 has a built-in procedure for plotting the graphs of functions in polar coordinates, as described beginning on p. 330 of the *Manual*. First, make sure that the calculator is in radian mode. Then specify POLAR type by pressing **POLAR** on the PLOT PTYPE menu. Then type in the equation  $r = f(\theta)$  and store it as the equation. Then set  $\theta$  to be the

independent variable, and DRAW it. (The symbol  $\theta$  is a right-shifted F in  $\alpha$  mode.) If you do not specify a range for the independent variable, the interval  $[0, 2\pi]$  is chosen automatically. To specify a range  $[c, d]$ , designate the list  $\{ \theta \ c \ d \}$  as the independent variable.

For example, to plot the graph of  $r = 2 \sin 4\theta$ , type

**PLOT** **PTYPE** **POLAR** ' R = 2 \* **SIN**  
4 \*  $\theta$  **ENTER** ( $\theta$  is  $\alpha$  right-shift F)

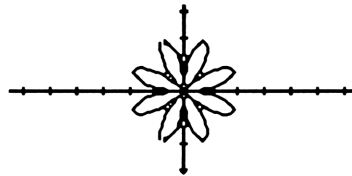
**STEQ** **PLOTR**  $\theta$  **INDEP**

**ERASE** **DRAW**.

1: 'R=2\*SIN(4\* $\theta$ )'  
**PLOT** **PTYPE** **NEW** **EQN** **TYPE** **OK**

Indep: ' $\theta$ '  
x: -6.5 6.5  
y: -3.1 3.2

**ERASE** **GRAPH** **AUTO** **WIND** **WIND** **INDEP**



## Chapter 9. Solid Analytic Geometry

### Points and Planes; Vector Methods

In three dimensions, a point has three coordinates, and can be represented readily to the HP calculator as a vector. That is, the point  $(x, y, z)$  can be given as the array  $[x \ y \ z]$ . A plane in space has an equation of the form  $ax + by + cz + d = 0$ , and can be represented to the calculator as the vector  $[a \ b \ c \ d]$ . These ideas are used in the following little programs.

#### The Plane on Three Points

$P3 \rightarrow \pi$  is a program that takes three points from the stack and returns the plane they determine.

```
« → U V W
  « U V - U W - CROSS → A
    « A 1 GET 'X' * A 2 GET 'Y' * + A 3 GET 'Z' * + A U DOT
  - 0 =
  »
»
```

checksum: # 35634d

For example, to find the plane containing the three points  $(2, 0, 0)$ ,  $(0, 3, 0)$ , and  $(0, 0, 5)$ , type

$[2, 0, 0]$ <input type="button" value="ENTER"/>	$[0, 3, 0]$ <input type="button" value="ENTER"/>	3:	$[2 \ 0 \ 0]$
$[0, 0, 5]$ <input type="button" value="ENTER"/>		2:	$[0 \ 3 \ 0]$
		1:	$[0 \ 0 \ 5]$
		$P3 \rightarrow \pi$ <input type="button" value="0.FTπ"/> <input type="button" value="P3→L"/> <input type="button" value="πππ→"/>	
$P3 \rightarrow \pi$ .		1:	$15X+10Y+6Z-30=0$
		$P3 \rightarrow \pi$ <input type="button" value="0.FTπ"/> <input type="button" value="P3→L"/> <input type="button" value="πππ→"/>	

If the result is ever  $0 = 0$ , it means that the three points given are collinear, and hence do not determine a plane.

#### Distance from a Point to a Plane

$D.PT\pi$  is a program that takes a point and a plane from the stack and returns the (shortest) distance between them. Both the point and plane are entered as vectors.

```
« → P PL
  « P OBJ→ DROP 1 4 →ARRY PL DOT ABS PL OBJ→ DROP DROP 3
  →ARRY ABS /
```

»  
»

checksum: # 48188d

For example, to find the distance between the point (1, 2, -1) and the plane  $3x + 2y - 4z - 7 = 0$ , type

[ 1 , 2 , 1 [CHS] [ENTER] [ 3 , 2 , 4  
[CHS] , 7 [CHS] [ENTER]

[D.PTπ].

2: [ 1 2 -1 ]  
1: [ 3 2 -4 -7 ]  
P3→T [0.0Tπ] P2→L [TπT→]  
1: .742781352709  
P3→T [0.0Tπ] P2→L [TπT→]

## Lines

A line in three dimensions can be represented most easily as a set of three linear parametric equations in the form

$$\begin{cases} x = x_0 + at \\ y = y_0 + bt \\ z = z_0 + ct \end{cases}$$

The following little programs create the parametric equations of lines.

### Line on Two Points

PS→L takes two points in space and gives the three parametric equations of the line they determine.

« → P1 P2  
« P1 P2 - → D  
« 'X' P1 1 GET D 1 GET 'T' \* + = 'Y' P1 2 GET D 2 GET  
'T' \* + = 'Z' P1 3 GET D 3 GET 'T' \* + =  
»  
»  
»

checksum: # 50865d

For example, to find the line on the points (2, 3, 1) and (5, 4, 0), type

[ 2 , 3 , 1 [ENTER] [ 5 , 4 , 0 [ENTER]

[PS→L].

2: [ 2 3 1 ]  
1: [ 5 4 0 ]  
P3→T [0.0Tπ] P2→L [TπT→]  
3: 'X=2-3\*T'  
2: 'Y=3-T'  
1: 'Z=1+T'  
P3→T [0.0Tπ] P2→L [TπT→]

**Line of Intersection of Two Planes**

$\pi\pi\rightarrow L$  takes two planes from the stack and returns the parametric equations of the line of intersection. The plane  $ax + by + cz + d = 0$  is entered as the vector  $[a \ b \ c \ d]$ .

```

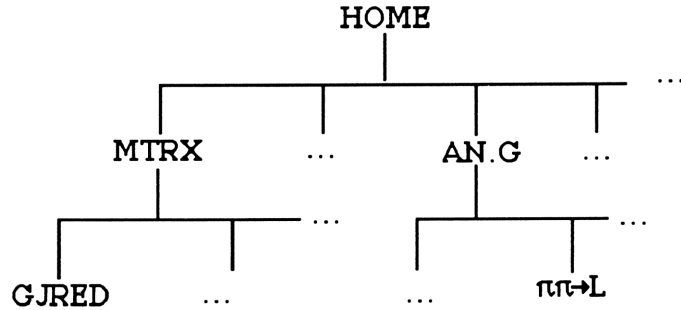
« → P Q
  « P OBJ→ DROP DROP 3 →ARRY Q ARRY→ DROP DROP 3 →ARRY CROSS
→ D
  « D ABS 0
    IF ==
      THEN "SAME OR PARALLEL"
      ELSE P OBJ→ DROP NEG Q OBJ→ DROP NEG ( 2 4 ) →ARRY
GJRED → A
  « 'A(1,1)' EVAL 0
    IF ==
      THEN 'X' D 1 GET 'T' * = 'Y' 'A(1,4)' EVAL D 2
GET 'T' * + = 'Z' 'A(2,4)' EVAL D 3 GET 'T' * + =
      ELSE 'X' 'A(1,4)' EVAL D 1 GET 'T' * + =
'A(2,2)' EVAL 0
      IF ==
      THEN 'Y' D 2 GET 'T' * = 'Z' 'A(2,4)' EVAL
D 3 GET 'T' * + =
      ELSE 'Y' 'A(2,4)' EVAL D 2 GET 'T' * + =
'Z' D 3 GET 'T' * =
      END
    END
  »
END
»
»
»
»

```

checksum: # 33318d

Notice that the above program uses the program GJRED, assuming it is in the same or a higher directory. If it is in a different subdirectory, then navigation directions to its location and back again must be given.

For example, if GJRED is in a subdirectory called MTRX and the program  $\pi\pi\rightarrow L$  is located in a subdirectory called AN.G, as in the diagram, then the command GJRED in the above program should be replaced with HOME MTRX GJRED HOME AN.G.



To use the program, for example, to find the line common to the planes  $3x + 2y + z - 6 = 0$  and  $x + 4y + 8z - 12 = 0$ , type

[ 3 , 2 , 1 , 6 ] [CHS] [ENTER] [ 1 , 4 ,  
8 , 12 ] [CHS] [ENTER]

[  $\pi\pi\rightarrow L$  ] .

```

2:      [ 3 2 1 -6 ]
1:      [ 1 4 8 -12 ]
PE->T 0.0T\T P->L \T\T->

4: [ [ 3 2 1 6 ] [ 1 ...
3: 'X=-.000000001+12*...
2:      'Y=3-23*T'
1:      'Z=10*T'
PE->T 0.0T\T P->L \T\T->
  
```

## Space Curves and Graphs

A curve in space is most easily represented parametrically, as

$$\begin{cases} x = f(t) \\ y = g(t) \\ z = h(t) \end{cases}, t \in [a, b].$$

To sketch the graph of a curve in space requires what is called a *perspective transformation* to make the graph appear as though we were actually looking at it. This in turn requires that we specify the point from which we are looking, and then perform the perspective transformation on each point we plot.

Here is a subdirectory TH.D.G that is an environment for plotting space curves (and surfaces, which we will talk about later). It does so by performing the perspective transformation on the three spacial dimensions, so that the picture created on the screen represents the object as seen when looking toward the origin from a specified viewpoint (VX, VY, VZ) in space. Means are included to allow for magnification by a scale factor S.

The menu under TH.D.G is

SPCRV	P.SRF	VSTO	SSTO	M	S
VX	VY	VZ	TRANS		

TRANS is a subroutine that performs the perspective transformation on a point of three-space. It uses the matrix M and the scale factor S that are supplied separately, as explained below. The input to TRANS is the set of coordinates x, y, z of a point in space.

The program is

```
« 1 4 →ARRY M SWAP * OBJ→ DROP DROP ROT ROT R→C S * SWAP INV
*
»
```

checksum: # 46002d

VX, VY, and VZ are the coordinates of the viewpoint, and are created by VSTO.

S is the scale factor, and is created by SSTO.

M is the matrix that performs most of the transformation, and is created by VSTO.

SSTO is just the program « 'S' STO » for storing the scale factor S. I find that a value of about 20 for S gets most of the picture on the screen, depending on the function and on the viewing point. You will have to experiment a little.

VSTO is a program for storing the coordinates of the viewpoint and creating the matrix M. It requires as input the coordinates VX, VY, and VZ (in that order). It is a good idea to select a viewpoint that will be outside the region of space in which the object to be viewed lies. The program is

```
« 'VZ' STO 'VY' STO 'VX' STO VX VY 2 →ARRY ABS 'D1' STO D1 VZ
2 →ARRY ABS 'D2' STO CLΣ VY NEG D1 / VX D1 / 0 0 4 →ARRY Σ+
VX VZ * NEG D1 D2 * / VY VZ * NEG D1 D2 * / D1 D2 / 0 4 →ARRY
Σ+ VX D2 / VY D2 / VZ D2 / D2 NEG 4 →ARRY NEG Σ+ 0 0 0 1 4
→ARRY Σ+ RCLΣ 'M' STO CLΣ { D1 D2 } PURGE
»
```

checksum: # 26746d

The commands involving Σ in the above program are found on the STAT menu.

P.SRF is a subdirectory for the plotting of surfaces, as will be explained later.

SPCRV is a subdirectory for plotting the graph of the three parametric equations

$$\begin{cases} x = f(t) \\ y = g(t) \\ z = h(t) \end{cases}, t \in [a, b]$$

in space. The menu for SPCRV is

XYZST	ABSTO	DRAGR	RCLGR	E.OLD	N
X	Y	Z	A	B	PPAR

PPAR contains the plot parameters. Choice of plot parameters is much less important in this setting than the viewpoint and the scale factor S, so the default plot parameters should be fine.

A and B are the limits on the parameter T, and are created by ABSTO.



X, Y, and Z are the functions f(t), g(t), and h(t), and are stored by XYZST.

N is the number of points that will be plotted, and must be stored separately.

E.OLD is a program for erasing the old picture if you don't want the next picture drawn on top of it, and is simply the program « ERASE ».

RCLGR is a program for recalling the last picture drawn, and is simply « GRAPH ».

DRAGR is the program that does the plotting. The program is

```
« B A - N / →NUM 'DT' STO A 'T' STO 1 N
  START X →NUM Y →NUM Z →NUM TRANS PIXON T DT + 'T' STO
  NEXT { T DT } PURGE GRAPH
```

»

checksum: # 22365d

ABSTO is the program « 'B' STO 'A' STO ». It takes the limits A and B of T from the stack and stores them.

XYZST takes the three functions f(t), g(t), and h(t) from the stack and stores them. The functions are entered in algebraic notation, using T as the variable. The program is just

```
« 'Z' STO 'Y' STO 'X' STO ».
```

To use this environment, suppose we wish to plot the graph of

$$\begin{cases} x = \cos t \\ y = 3 \sin t \\ z = t/5 \end{cases}, t \in [0, 2\pi].$$

We must first choose a viewpoint, say (20, 10, 15). This we do by entering the subdirectory TH.D.G and typing

20 [ENTER] 10 [ENTER] 15

```
2: 20
1: 10
15
SPCRV P.SCF VSTO SSTO N E
```

[VSTO]. We must also choose a scale constant S, and 40 is probably a good choice. Type 40 [SSTO]. Then enter the subdirectory SPCRV. If we want to plot 100 points, we type 100 [ENTER] 'N [STO]. To enter the parametric equations, we type

```
' C O S ( T [ENTER] ' 3 * S I N ( T
[ENTER] ' T / 5 [ENTER]
```

```
3: 'COS(T)'
2: '3*SIN(T)'
1: 'T/5'
XYZST RCLGR DRAGR RCLG E.OLD EXIT
```

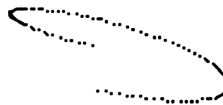
[XYZST]. The interval [0, 2π] is most easily entered by typing

0 [ENTER] 6.29

```
1: 0
6.29
XYZST RCLGR DRAGR RCLG E.OLD EXIT
```

**ABSTO**. Finally, plot the graph by pressing

**DRAGR**.



## Parametric Surfaces

The various surfaces mentioned in calculus can all be graphed on the HP 48. This is often a slow and tedious job, even for the calculator, because so much computation is going on; some even suggest that it be left to larger computers entirely. However, it is fun to see what the calculator can do, so we present the material here for you to use if you want to.

We will describe all surfaces in terms of parametric surfaces. By way of definition, a parametric surface is the graph of the parametric equations

$$\begin{cases} x = f(u,v) \\ y = g(u,v) \\ z = h(u,v) \end{cases}, u \in [a, b], v \in [c, d].$$

If a value of  $v$  is chosen and  $v$  is fixed at that value, then the equations become parametric equations in terms of the single parameter  $u$ , whose graph is a space curve; this curve is called a  $v$ -*curve* of the surface, and the set of all  $v$ -curves is called the  $v$ -*net* of the surface. Similarly, if the value of  $u$  is fixed, we get a  $u$ -*curve*, and the set of all  $u$ -curves is called the  $u$ -*net* of the surface.

By plotting a few curves of each net, a "wire-mesh" model of the surface is obtained; that is what we will mean by the graph of the surface. Before discussing the environment for plotting parametric surfaces on the HP 48, we will describe some common surfaces in terms of parametric equations, and give a few examples.

### Planes

A plane in space has an equation of the form  $ax + by + cz + d = 0$ . If the  $z$ -term is present ( $c \neq 0$ ), then we may solve for  $z$  and get  $z = -\frac{ax + by + d}{c}$ . Therefore the parametric equations

$$\begin{cases} x = u \\ y = v \\ z = -\frac{au + bv + d}{c} \end{cases}$$

give us the plane. The  $u$ - and  $v$ -nets are lines in that plane parallel to the  $xz$ - and  $yz$ -planes.

If the  $z$ -term is absent, so that the equation of the plane is  $ax + by + d = 0$ , then the plane is vertical (parallel to the  $z$ -axis). If the  $y$ -term is present ( $b \neq 0$ ), we may solve for  $y$  and get  $y = -\frac{ax + d}{b}$ , and the parametric equations

$$\begin{cases} x = u \\ y = -\frac{au + d}{b} \\ z = v \end{cases}$$

give us the plane. The  $u$ -net consists of vertical lines, and the  $v$ -net consists of horizontal lines.

If both the  $y$ - and  $z$ -terms are absent, then the plane is of the form  $x = k$ , and the parametric equations

$$\begin{cases} x = k \\ y = u \\ z = v \end{cases}$$

give us the plane. The  $u$ - and  $v$ -nets are vertical and horizontal lines again.

### Graphs of Functions of Two Variables

The graph of a function  $z = f(x, y)$  of two variables is a surface in space. It is representable as a parametric surface in the form

$$\begin{cases} x = u \\ y = v \\ z = f(u, v) \end{cases}$$

The  $u$ -curves of this representation are traces of the surface in planes parallel to the  $xz$ -plane, and the  $v$ -curves are traces in planes parallel to the  $yz$ -plane.

### Quadric Surfaces

The various quadric surfaces can be represented as parametric surfaces using trigonometric functions. Consider the following examples.

The sphere of radius  $r$  centered at the origin can be described at each point  $P$  by specifying two angles, an angle  $u$  in the  $xy$ -plane from the positive  $x$ -axis (corresponding to  $\theta$  in cylindrical coordinates) and an angle  $v$  of elevation from the  $xy$ -plane to the point  $P$ . By computing the coordinates of  $P$ , we find the parametric equations of the sphere to be

$$\begin{cases} x = r \cos u \cos v \\ y = r \sin u \cos v \\ z = r \sin v \end{cases}, u \in [0, 2\pi], v \in [-\frac{\pi}{2}, \frac{\pi}{2}].$$

The  $u$ -net consists of the circles of longitude of the sphere, and the  $v$ -net consists of the circles of latitude.

A portion of the sphere can be obtained by restricting  $u$  or  $v$ . For example, requiring  $v \in [0, \frac{\pi}{2}]$  gives us the upper hemisphere.

The ellipsoid with semi-axes  $a$ ,  $b$ , and  $c$ , centered at the origin, is a slight generalization of the sphere, and has parametric equations

$$\begin{cases} x = a \cos u \cos v \\ y = b \sin u \cos v \\ z = c \sin v \end{cases}, \quad u \in [0, 2\pi], v \in [-\frac{\pi}{2}, \frac{\pi}{2}].$$

The elliptic paraboloid  $z = \frac{x^2}{a^2} + \frac{y^2}{b^2}$  can be represented as for other functions of two variables if you want the  $u$ - and  $v$ -nets to be vertical plane sections. If you want horizontal sections, the parametrization

$$\begin{cases} x = au \cos v \\ y = bu \sin v \\ z = u^2 \end{cases}, \quad u \in [0, \infty), v \in [0, 2\pi]$$

has as its  $u$ -net the set of horizontal plane sections.

A hyperboloid of one sheet can be represented as

$$\begin{cases} x = a \cosh u \cos v \\ y = b \cosh u \sin v \\ z = c \sinh u \end{cases}, \quad u \in [0, \infty), v \in [0, 2\pi].$$

The  $v$ -net is the set of sections by planes containing the  $z$ -axis, and the  $u$ -net is the set of horizontal plane sections.

Most of the other quadric surfaces can be represented in similar fashion, perhaps by interchanging  $x$ ,  $y$ , and  $z$ , or can be represented as functions of two variables.

### Surfaces of Revolution

Suppose the curve  $z = f(y)$ ,  $x = 0$ ,  $y \in [c, d]$  is rotated about the  $z$ -axis to obtain a surface of revolution. If  $u$  represents the angle of rotation at any point, the parametric equations of the surface are

$$\begin{cases} x = v \cos u \\ y = v \sin u \\ z = f(v) \end{cases}, \quad u \in [0, 2\pi], v \in [c, d].$$

The  $v$ -net consists of circles traced by points on the original curve, and the  $u$ -net consists of section by half-planes containing the  $z$ -axis.

Suppose the circle of radius  $a$  in the  $yz$ -plane, centered at the point  $(0, b, 0)$ , is revolved about the  $z$ -axis to obtain a torus. If  $u$  is the angle of rotation, and  $v$  is the angle of elevation from the center of the rotated circle to a point on the surface, then the parametric equations of the torus are

$$\begin{cases} x = (b - a \cos v) \cos u \\ y = (b - a \cos v) \sin u \\ z = a \sin v \end{cases}, u, v \in [0, 2\pi].$$

The u-net consists of circular cross-sections by half-planes containing the z-axis, and the v-net consists of horizontal cross-sections.

Other surfaces of revolution can be represented in a similar fashion.

### Graphing Parametric Surfaces

Here is an environment for the graphing of parametric surfaces on the HP 48. It fits in the P.SRF subdirectory on the TH.D.G menu mentioned earlier. The complete menu under P.SRF is

XYZST	ABSTO	CDSTO	NUVST	DRAGR	RCLGR
NU	NV	U.N.V	X	Y	Z
A	B	C	D	USCR	VSCR
SCR	PPAR				

PPAR contains the plot parameters (default parameters are best).

SCR contains the entire plot of the surface.

VSCR and USCR contain the screens of the v-net alone and the u-net alone.

C and D are the limits of the v-curves; A and B are the limits of the u-curves.

X, Y, and Z are containers for the functions that define the surface.

U.N.V is for putting the u-net and the v-net graphs together into the display. It is the program

```
« USCR VSCR + DUP 'SCR' STO PICT STO GRAPH »
checksum: # 14776d
```

NV is the number of curves in the v-net that are to be plotted. NU is the number of curves in the u-net.

RCLGR is a subdirectory containing the three commands RCLS, RCLU, and RCLV.

RCLV is for recalling the v-net. It is the program

```
« VSCR PICT STO UPDIR GRAPH ».
```

RCLU puts the graph of the u-net on the screen. It is the program

```
« USCR PICT STO UPDIR GRAPH ».
```

RCLS puts the graph of the surface (both the u-net and the v-net) on the screen. It is the program « UPDIR U.N.V ».

This completes the description of RCLGR.

DRAGR is a subdirectory that contains the commands DRAS, DRAU, and DRAV, and the container PPAR.

PPAR contains the plot parameters (default parameters are fine).

DRAV sketches the v-net; here is the program:

```
« D C - NU / 'DV' STO B A - NV / 'DU' STO A 'U' STO C 'V' STO
ERASE ( # 0d # 0d ) PVIEW 1 NU 1 +
  START X →NUM Y →NUM Z →NUM TRANS 1 NV
  START DU 'U' STO+ X →NUM Y →NUM Z →NUM TRANS DUP 3
ROLLD LINE
  NEXT DROP DV 'V' STO+ A 'U' STO
  NEXT ( U V DU DV ) PURGE PICT RCL UPDIR 'VSCR' STO
»
```

checksum: # 32157d

DRAU sketches the u-net. Here is the program:

```
« B A - NV / 'DU' STO D C - NU / 'DV' STO C 'V' STO A 'U' STO
ERASE ( # 0d # 0d ) PVIEW 1 NV 1 +
  START X →NUM Y →NUM Z →NUM TRANS 1 NU
  START DV 'V' STO+ X →NUM Y →NUM Z →NUM TRANS DUP 3
ROLLD LINE
  NEXT DROP DU 'U' STO+ C 'V' STO
  NEXT ( U V DU DV ) PURGE PICT RCL UPDIR 'USCR' STO
»
```

checksum: # 13721d

DRAS draws both the u-net and the v-net and puts them together, and is the program

```
« DRAU DRAGR DRAV U.N.V ».
```

This completes the description of DRAGR.

CDSTO is for storing the values C and D, and is the program « 'D' STO 'C' STO ». It takes the values for C and D from the stack.

ABSTO is just like CDSTO.

XYZST is for storing the functions that define the surface. It is the program

```
« 'Z' STO 'Y' STO 'X' STO »,
```

and takes the three functions from the stack. The functions should be entered in algebraic form, using U and V as the parameters.



## Velocity and Acceleration

For example, the velocity and acceleration of a vector (position) function can be computed by the following little program for differentiating a vector function, called VDIF. It takes a list of three parametric functions, written in terms of the parameter T, from the stack, differentiates them, and returns both the function and its derivative to the stack.

```
« → V
  « V V 1 GET 'T' ⋔ V 2 GET 'T' ⋔ V 3 GET 'T' ⋔ 3 →LIST
  »
»
```

checksum: # 22898d

For example, given the vector function  $\mathbf{r}(t) = [\sin t, t^2, 2 - 3t]$ , we compute the velocity by differentiating. We first construct the list for  $\mathbf{r}$ , by typing

```
'SIN(T) [ENTER] 'T^2 [ENTER]
'2-3*T [ENTER] 3
```

```
3: 'SIN(T)'
2: 'T^2'
1: '2-3*T'
3
```

→LIST (on the PRG OBJ menu).

```
1: { 'SIN(T)' 'T^2' '2-3*T' }
DEJ ← EOP ← NAME ← LIST ← TR ← TMS
```

Then we differentiate by pressing

VDIF.

```
2: { 'SIN(T)' 'T^2' '2-3*T' }
1: { 'COS(T)' '2*T' '-3' }
»
```

We calculate the acceleration by again pressing

VDIF.

```
3: { 'SIN(T)' 'T^2' '2-3*T' }
2: { 'COS(T)' '2*T' '-3' }
1: { '-SIN(T)' '2' '0' }
»
```

To evaluate a vector function at a given value of the parameter, the following function is useful. It is called VVAL. It assumes that the vector function is in level 2 and the value of T is in level 1, and it returns both the function and its value to the stack.

```
« 'T' STO DUP → V
  « V 1 GET →NUM V 2 GET →NUM V 3 GET →NUM 3 →LIST 'T' PURGE
  »
»
```

checksum: # 23040d

For example, given the vector function above, to evaluate it at  $t = 5$ , we create the list of parametric functions as above, and then type



5

```
1: ( 'SIN(T)' 'T^2' '2
    -3*T' )
```

5

```
VDIF VVAL CRVT SIND VFWL DIFGE
```

VVAL.

```
2: ( 'SIN(T)' 'T^2' '
1: ( -.958924274663 25
    -13 )
```

```
VDIF VVAL CRVT SIND VFWL DIFGE
```

## Curvature

The following program, called CRVT, computes the curvature of a parametric curve in three-space, using the formula

$$\kappa = \frac{\|\mathbf{r}' \times \mathbf{r}''\|}{\|\mathbf{r}'\|^3}.$$

Because the curvature in symbolic form is usually very complicated, this program returns the value of the curvature at a point. The program can be modified, by writing routines for the cross product and magnitude of vector functions given as lists, to return the symbolic expression if preferred. This program takes a list of three parametric equations (the position function  $\mathbf{r}$ ) from level 2 and the value of  $T$  from level 1 and returns the value of  $\kappa$ .

```
< → R T
```

```
  < R VDIF T VVAL OBJ→ →ARRY 'RP' STO VDIF T VVAL OBJ→ →ARRY
  RP SWAP CROSS ABS RP ABS 3 ^ / 'RP' PURGE SWAP DROP SWAP DROP
  »
```

```
»
```

checksum: # 44326d.

For example, to find the curvature of the curve  $\mathbf{r}(t) = [\sin t, t^2, 2 - 3t]$  at the point  $t = 0$ , we type

```
'SIN(T) [ENTER] 'T^2 [ENTER]
'2-3*T [ENTER] 3 [→LIST] 0
```

CRVT.

```
1: ( 'SIN(T)' 'T^2' '2
    -3*T' )
```

0

```
VDIF VVAL CRVT SIND VFWL DIFGE
```

```
2: ( 'SIN(T)' 'T^2' '
1: ( .2
```

```
VDIF VVAL CRVT SIND VFWL DIFGE
```

## Cylindrical Coordinates

Since the rectangular coordinates  $(x, y, z)$  and the cylindrical coordinates  $[r, \theta, z]$  are related in the same way as rectangular and polar coordinates in the first two variables, the HP 48 will transform from rectangular to cylindrical coordinates. We use the command POLAR.

The HP 48 will also handle spherical coordinates, as explained on p. 171 of the *Manual*.



## **Chapter 10. Partial Differentiation**

The HP 48 can be used to accomplish several things in the calculus of several variables, including graphing functions of two variables, creating contour maps, computing partial derivatives, and constructing gradients and directional derivatives.

### **Functions of Two Variables**

#### **Graphs**

The graph of the function  $z = f(x, y)$  is obtained by using the parametric graph unit of Chapter 10, with the parametric equations

$$\begin{cases} x = u \\ y = v \\ z = f(u, v) \end{cases}.$$

To get the graph of the portion of the surface above the rectangle  $[a, b] \times [c, d]$ , set  $u$  to run from  $a$  to  $b$  and  $v$  to run from  $c$  to  $d$ . You will get a wire mesh representation of the surface.

It is also possible to get a "hidden line" version of the graph, in which parts of the surface hidden by other parts in front of them are not shown, but it is an intensive job, perhaps best left to a larger computer.

#### **Contour Maps**

Contour maps of the function  $z = f(x, y)$  can be created on the HP 48 by using IMP.G from Chapter 8. To plot the  $c$ -level curve of  $z = f(x, y)$ , plot the implicit graph of  $f(x, y) - c = 0$ .

#### **Partial Derivatives**

Since the independent variable must be specified for the  $\frac{\partial}{\partial}$  command to work, partial differentiation is just the same as differentiation, but with other symbols around. The procedure for getting partial derivatives is identical to that for differentiating functions of one variable.

Programs can be developed for getting higher-order partial derivatives, mixed partials, and so forth. Creation of such programs is left to the interested reader.

#### **Two-Variable Newton's Method**

Given a system (not usually linear) of two equations in two unknowns

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases},$$

there is a two-variable version of Newton's method that takes an estimate of a solution and returns (hopefully) a better estimate. Without discussing the theory, which can be found in elementary numerical analysis books, we give here a set of programs for the two-variable Newton's method. On the calculator, it behaves just like the Newton's method discussed in Chapter 4.

First comes the program FGSTO:

```
« DUP 'X' ⋔ 'GX' STO DUP 'Y' ⋔ 'GY' STO 'G' STO DUP 'X' ⋔
'FX' STO DUP 'Y' ⋔ 'FY' STO 'F' STO
»
```

checksum: # 23092d

FGSTO takes the two functions  $f(x, y)$  and  $g(x, y)$  from the stack and stores them and their partial derivatives for later use.

Then comes the program NEWT2:

```
« 'Y' STO 'X' STO X Y F GY * G FY * - →NUM FX GY * FY GX * -
→NUM DUP 'J' STO / X SWAP - FX G * F GX * - →NUM J / Y SWAP -
'Y' STO 'X' STO X Y ( X Y J ) PURGE
»
```

checksum: # 9370d

NEWT2 takes the estimate  $x_1$  and  $y_1$  from the stack and returns a new estimate  $x_2$  and  $y_2$ .

As an example, to solve the system of equations

$$\begin{cases} (x - 50)^2 - 25y^2 = 100 \\ (y - 40)^2 - x^2 = 400 \end{cases},$$

we type

```
' ( X - 50 ) ^ 2 - 25 * Y ^ 2 - 100
[ENTER] ' ( Y - 40 ) ^ 2 - X ^ 2 -
400 [ENTER]
```

```
2: '(X-50)^2-25*Y^2-100
1: '(Y-40)^2-X^2-400
[REST] [NEXT] [NNEXT] [F] [G] [F']
```

[FGSTO]. Then we enter an estimate; if (20, 10) is our guess, we type

```
20 [ENTER] 10
```

```
1: 20
10
[REST] [NEXT] [NNEXT] [F] [G] [F']
```

[NEWT2].

```
4: 20
3: 10
2: 29.2682926829
1: 5.48780487805
[REST] [NEXT] [NNEXT] [F] [G] [F']
```

Press **NEWT2** several times.

```
4: 29.2682926829
3: 5.48780487805
2: 30.118305159
1: 3.81740732881
PSET NEWT NNWT F G F'
```

```
4: 30.118305159
3: 3.81740732881
2: 30.7436329949
1: 3.3254591324
PSET NEWT NNWT F G F'
```

```
4: 30.7436329949
3: 3.3254591324
2: 30.7963882654
1: 3.27920362187
PSET NEWT NNWT F G F'
```

We can have the machine press **NEWT2** for us several times, by using a program such as NNWT2:

```
« → N
  « 1 N
    START NEWT2
    NEXT
  »
```

This program takes the estimated solution from levels 3 and 2 and the number of repetitions from level 1 and returns that many successive estimates.

These programs can be organized into a subdirectory, perhaps called NWT2. The complete menu might be

```
FGSTO  NEWT2  NNWT2  F      G      FX
FY      GX      GY
```

## Gradients

A gradient of a function of three variables is just a vector field in which the components are the partial derivatives. Here is a simple program for computing the gradient of a function of three variables, and presenting it as a list of functions. It is called GRADI.

```
« → F
  « F 'X' ∂ F 'Y' ∂ F 'Z' ∂ 3 →LIST
  »
»
```

checksum: # 50590d

This program takes a function of three variables, written in terms of X, Y, and Z, from the stack and returns the gradient as a list of the three partial derivatives.

For example, to find the gradient of the function  $F(x, y, z) = x^2y - 3z$ , type

```
'X ^ 2 * Y - 3 * Z' [ENTER]
[GRADI].
```

1: 'X^2\*Y-3\*Z'  
 1: { '2\*X\*Y' 'X^2' -3

Here is a little program, called VFVAL, for evaluating a vector field of three variables at a point. It takes the vector field, written as a list, from level 2 and the point, written as a vector, from level 1. It returns the value of the field as a vector.

```
< OBJ→ DROP 'Z' STO 'Y' STO 'X' STO → F
  < F 1 GET →NUM F 2 GET →NUM F 3 GET →NUM 3 →ARRY { X Y Z }
  PURGE
  »
  »
```

checksum: # 15924d

For example, to evaluate the vector field  $[2xy, x^2, 2z]$  at the point  $(1, 5, 3)$ , type

```
'2 * X * Y' [ENTER] 'X ^ 2' [ENTER]
'2 * Z' [ENTER] 3 →LIST
[1, 5, 3] [ENTER]
[VFVAL].
```

1: { '2\*X\*Y' 'X^2' '2\*Z' }  
 2: { '2\*X\*Y' 'X^2' '2\*Z' }  
 1: [ 1 5 3 ]  
 1: [ 10 1 6 ]

## Directional Derivatives

Vector-handling routines on the HP 48 make the computation of the directional derivative easy to program. To compute the directional derivative of a function  $f$  at a point  $P$  in the direction of a vector  $v$ , we form the dot product of the gradient of  $f$  at  $P$  with a unit vector in the direction of  $v$ . The following program, called DIRDE, takes the function  $f$  from level 3, the point  $P$  (written as a vector) from level 2, and the vector  $v$  from level 1, and returns the directional derivative.

```
< → F P V
  < F GRADI P VFVAL V V ABS / DOT
  »
  »
```

checksum: # 31755d

For example, to find the directional derivative of the function  $f(x, y, z) = x^2y - 3z$  at the point  $(1, 3, 5)$  in the direction of  $\mathbf{v} = [1, 2, 1]$ , type

'X ^ 2 \* Y - 3 \* Z' [ENTER] [1, 3,  
5 [ENTER] [1, 2, 1 [ENTER]

```
3:      'X^2*Y-3*Z'
2:      [ 1 3 5 ]
1:      [ 1 2 1 ]
      NOIF  WNL  DENT  SEAR0  WFWNL  DIRDE
```

[DIRDE].

```
1:      2.04124145232
      NOIF  WNL  DENT  SEAR0  WFWNL  DIRDE
```

This same program will also work for functions of two variables; just put in zero for the third component. For example, to find the directional derivative of  $f(x, y) = x^2 - y^2$  at the point  $(4, 1)$  in the direction of  $\mathbf{v} = [1, 1]$ , type

'X ^ 2 - Y ^ 2' [ENTER] [4, 1, 0  
[ENTER] [1, 1, 0 [ENTER]

```
3:      'X^2-Y^2'
2:      [ 4 1 0 ]
1:      [ 1 1 0 ]
      NOIF  WNL  DENT  SEAR0  WFWNL  DIRDE
```

[DIRDE].

```
1:      4.24264068713
      NOIF  WNL  DENT  SEAR0  WFWNL  DIRDE
```





## **Chapter 11. Path Integrals and Multiple Integrals**

Capabilities of the HP 48 enable the evaluation of path integrals and multiple integrals in a very efficient manner. Here are some programs that do these things, taking much of the drudgery out of applications.

### **Path Integrals**

The path integral of the vector field  $\mathbf{F} = \langle M(x, y, z), N(x, y, z), P(x, y, z) \rangle$  along the curve C:

$\mathbf{r}(t) = \langle x(t), y(t), z(t) \rangle$ ,  $t \in [a, b]$ , is  $\int_C \mathbf{F} \cdot d\mathbf{r} = \int_a^b \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt$ . An environment for the evaluation of such path integrals is presented here.

Start with a subdirectory for path integrals, perhaps called P.INT, which contains the following programs and containers:

FSTO	RSTO	ABSTO	PINT	IERR	M
N	P	X	Y	Z	A
B	XP	YP	ZP		

- XP, YP, and ZP are storage containers for the components of the derivative  $\mathbf{r}'$ .
- A and B are storage containers for a and b.
- X, Y, and Z are storage containers for the components of  $\mathbf{r}$ .
- M, N, and P are storage containers for the components of  $\mathbf{F}$ .
- IERR is a container for the error of integration.
- PINT is the program that does the calculation. The program is

```
« RCLF A B M EVAL XP * N EVAL YP * + P EVAL ZP * + 'T' 5 FIX
J →NUM SWAP STOF
»
```

checksum: # 15922d

This program evaluates  $\mathbf{F}$  at  $\mathbf{r}(t)$ , forms the dot product with  $\mathbf{r}'(t)$ , and then uses the machine's built-in numerical integration routine to evaluate the integral. The number .00001 =  $10^{-5}$  is specified as the error tolerance; to change the tolerance, change the number 5 in the program. The result of the program is two numbers: the value of the integral is placed on the stack and the maximum error is stored in IERR.

ABSTO is the program « 'B' STO 'A' STO » for storing the limits a and b of the parameter, describing the curve C. These become the limits of the integral that is evaluated.

RSTO is the program

```
« DUP 'T' ⋈ 'ZP' STO 'Z' STO DUP 'T' ⋈ 'YP' STO 'Y' STO DUP
'T' ⋈ 'XP' STO 'X' STO
»
```

checksum: # 8136d

for storing the vector function  $\mathbf{r}$  of which the curve  $C$  is the graph. The program also computes and stores the derivative  $\mathbf{r}'$ . The components  $X$ ,  $Y$ , and  $Z$  of  $\mathbf{r}$  should be entered onto the stack in that order, using  $T$  as the parameter.

FSTO is the program « 'P' STO 'N' STO 'M' STO » for storing the three components of the vector field  $\mathbf{F}$ . The components  $M$ ,  $N$ , and  $P$  should be entered onto the stack in that order, using variables  $X$ ,  $Y$ , and  $Z$ .

To illustrate, suppose we wish to find the path integral of  $\mathbf{F}(x, y, z) = \langle x + y + z, 2x - y - z, x - y + 3z \rangle$  over the circle  $C: \mathbf{r}(t) = \langle \cos t, \sin t, 0 \rangle$ ,  $t \in [0, 2\pi]$ . We type

'X + Y + Z' [ENTER] '2 * X - Y - Z'	3: 'X+Y+Z'
[ENTER] 'X - Y + 3 * Z' [ENTER]	2: '2*X-Y-Z'
	1: 'X-Y+3*Z'
	[FSTO] [FSTO] [FSTO] [PINT] [IERR] [M]
[FSTO]	3: 'COS(T)'
'COS(T)' [ENTER] 'SIN'	2: 'SIN(T)'
(T [ENTER] 0 [ENTER]	1: 0
	[FSTO] [FSTO] [FSTO] [PINT] [IERR] [M]
[RSTO]	2: 0
0 [ENTER] $\pi$ [ENTER] 2 * [→NUM]	1: 6.28318530718
	[FSTO] [FSTO] [FSTO] [PINT] [IERR] [M]
[ABSTO] [PINT].	1: 3.14159293912
	[FSTO] [FSTO] [FSTO] [PINT] [IERR] [M]
Press [IERR] to see the possible error.	2: 3.14159293912
	1: 7.49530816891E-5
	[FSTO] [FSTO] [FSTO] [PINT] [IERR] [M]

## Numerical Multiple Integration

The most straightforward approach to numerical integration for multiple integrals is to express them as iterated integrals and then use a nested Simpson's rule.

The version of Simpson's rule developed in Chapter 6 has quite a bit of inefficiency built into it, so we will first give a streamlined version of Simpson's rule for a definite (simple) integral, then for double iterated integrals, and finally for triple iterated integrals.

It is best to organize these programs into three subdirectories of a single directory M.INT (for Multiple INTeGration). The three subdirectories could be called INT1, INT2, and INT3, or maybe S.INT, D.INT, and T.INT, for simple, double, and triple integrals, respectively. We will choose the former.

## Simple Integrals

In the subdirectory INT1, there are several programs and storage containers for evaluating integrals of the form

$$\int_a^b f(x) dx.$$

The complete menu under INT1 is:

FABST	NSTO	SIMP1	F	N	A
B	H				

H is a container for storing the step size.

A and B are containers for storing the limits of the integral.

N is a container for storing half the number of subintervals. This number N is the same as the number N in Chapter 7; in Simpson's rule, the number of subintervals is 2N, since midpoints of the major subintervals are used, too.

F is a container for storing the integrand  $f(x)$ .

SIMP1 is the fast version of Simpson's rule for the integral, and is the program

```
« A 'X' STO F EVAL 1 N 2 * 1 -
  FOR I H X + 'X' STO F EVAL 2 * DUP 'T' STO + I 2 / DUP IP
    IF ≠
      THEN T +
    END
  NEXT B 'X' STO F EVAL + H * 3 / ( X T ) PURGE
»
```

checksum: # 22096d

NSTO is the program

```
« 'N' STO B A - N 2 * / 'H' STO »,
```

checksum: # 48933d

which not only stores N, half the number of subintervals involved, but also computes H, the step size.

FABST is the program « 'B' STO 'A' STO 'F' STO » for storing the integrand  $f(x)$  and the limits a and b. The name of the program reminds you to enter f, then a, and finally b. The function f should be written in algebraic form in terms of the variable X.

To illustrate the use of this environment, suppose we wish to evaluate the integral

$$\int_1^2 \frac{1}{x} dx, \text{ using } N = 4. \text{ We would type the following:}$$

```
' 1 / X [ENTER] 1 [ENTER] 2
2:
1:
2:
FABST NSTO SMP1 F N H
1: .693154530663
FABST NSTO SMP1 F N H
```

## Double Integrals

The subdirectory INT2 consists of programs and containers for evaluating double integrals of the form

$$\int_a^b \int_{c(x)}^{d(x)} f(x, y) dy dx.$$

The complete menu under INT2 is

FABST	CDSTO	MNST	SMP2	F	M
N	A	B	CX	DX	KSTO
SMP1	H				

H is a container for storing the "outer" stepsize.

SMP1 is the program

```
« C 'Y' STO F EVAL 1 N 2 * 1 -
  FOR J K Y + 'Y' STO F EVAL 2 * DUP 'T' STO + J 2 / DUP IP
    IF ≠
      THEN T +
    END
  NEXT D 'Y' STO F EVAL + ( Y T ) PURGE
»
```

checksum: # 60313d

KSTO is the program

```
« 'X' STO DX EVAL DUP 'D' STO CX EVAL DUP 'C' STO - N 2 * /
'K' STO
»
```

checksum: # 31931d

A, B, CX, and DX are containers for storing the limits a, b, c(x), and d(x).

M and N are containers for storing M and N, half the numbers of intervals in the inner and outer subdivisions.

F is a container for storing the integrand f(x, y).

SMP2 is the main program:

```
« A KSTO SMP1 1 M 2 * 1 -
  FOR I H X + KSTO SMP1 2 * DUP 'T' STO + I 2 / DUP IP
  IF ≠
  THEN T +
  END
  NEXT B KSTO SMP1 + H * 3 / { X T C D K } PURGE
»
```

checksum: # 2151d

MNST is the program

```
« 'N' STO 'M' STO B A - M 2 * / 'H' STO »,
```

checksum: # 56443d

for storing the numbers M and N, there being 2M subintervals in the X-direction and 2N subintervals in the Y-direction. It also computes the step size H in the X-direction.

CDSTO is the program « 'DX' STO 'CX' STO », for storing the limits c(x) and d(x). They are to be entered in algebraic form in terms of the variable X, and in that order.

FABST is the program « 'B' STO 'A' STO 'F' STO », for storing the integrand and the first two limits. The function f(x, y) should be entered in algebraic form, using the variables X and Y.

As an illustration of the use of this environment, suppose that we wish to estimate the

iterated integral  $\int_{0.1}^{0.5} \int_{x^3}^{x^2} e^{y/x} dy dx$  with  $M = N = 5$ . We would type the following:

'E X P ( Y / X [ENTER] .1 [ENTER] .5

```
2:      'EXP(Y/X)'
1:      .1
.5
PRST CDSTO MNST SMP2 F M
```

[FABST]  
'X ^ 3 [ENTER] 'X ^ 2 [ENTER]

```
2:      'X^3'
1:      'X^2'
PRST CDSTO MNST SMP2 F M
```

[CDSTO]  
5 [ENTER] [ENTER]

```
2:      5
1:      5
PRST CDSTO MNST SMP2 F M
```

[MNST] [SMP2].

```
1:      3.33054612819E-2
PRST CDSTO MNST SMP2 F M
```

### Triple Integrals

The subdirectory INT3 contains programs and containers for evaluating triple integrals of the form

$$\int_a^b \int_{c(x)}^{d(x)} \int_{e(x,y)}^{g(x,y)} f(x, y, z) dz dy dx.$$

The complete menu under INT3 is as follows:

FABST	CDEGS	MNPS	SMP3	F	M
N	P	A	B	CX	DX
EXY	GXY	KSTO	SMP2	LSTO	SIMP1
H					

H is a container for storing the stepsize H.

SMP1 is the program

```
« E 'Z' STO F EVAL 1 P 2 * 1 -
  FOR U L Z + 'Z' STO F EVAL 2 * DUP 'T' STO + U 2 / DUP IP
    IF ≠
      THEN T +
    END
  NEXT G 'Z' STO F EVAL + L * 3 / { Z T } PURGE
»
```

checksum: # 31372d

LSTO is the program

```
« 'Y' STO GXY EVAL DUP 'G' STO EXY EVAL DUP 'E' STO - P 2 * /
'L' STO
»
```

checksum: # 38671d

SMP2 is the program

```
« C LSTO SMP1 1 N 2 * 1 -
  FOR J K Y + LSTO SMP1 2 * DUP 'T' STO + J 2 / DUP IP
    IF ≠
      THEN T +
    END
  NEXT D LSTO SMP1 + K * 3 / { Y T } PURGE
»
```

checksum: # 8655d

KSTO is the program

```
« 'X' STO DX EVAL DUP 'D' STO CX EVAL DUP 'C' STO - N 2 * /
'K' STO
»
```

checksum: # 31931d

A, B, CX, DX, EXY, and GXY are containers for storing the limits of the integrals.

M, N, and P are containers for storing the numbers M, N, and P.

F is a container for storing the integrand  $f(x, y, z)$ .

SMP3 is the main program:

```
« A KSTO SMP2 1 M 2 * 1 -
  FOR I H X + KSTO SMP2 2 * DUP 'T' STO + I 2 / DUP IP
    IF ≠
      THEN T +
    END
  NEXT B KSTO SMP2 + H * 3 / { X T C D E G K L } PURGE
»
```

checksum: # 18400d

MNPS is the program

```
« 'P' STO 'N' STO 'M' STO B A - M 2 * / 'H' STO »
```

checksum: # 40817d

for storing the numbers M, N, and P that determine the numbers of subintervals in each direction, and also computing the outermost stepsize H.

CDEGS is the program « 'GXY' STO 'EXY' STO 'DX' STO 'CX' STO » for storing the limits  $c(x)$ ,  $d(x)$ ,  $e(x, y)$ , and  $g(x, y)$ . These functions should be entered in that order, using X and Y as the variables.

FABST is the program « 'B' STO 'A' STO 'F' STO », for storing the integrand  $f(x, y, z)$  and the limits a and b. The function f should be entered in algebraic form, using the variables X, Y, and Z.

As an illustration in the use of this environment, suppose we wish to evaluate the integral

$\int_0^1 \int_x^{x^2} \int_{xy}^{xy^2} (1 + xyz) dz dy dx$ , using  $M = N = P = 4$ . We would type

'1 + X \* Y \* Z' [ENTER] 0 [ENTER] 1

```
2:      '1+X*Y*Z'
1:      0
1:
FABST CDEGS MNPS SMP3 F M
```



FABST  
'X [ENTER] 'X ^ 2 [ENTER] 'X \* Y  
[ENTER] 'X \* Y ^ 2 [ENTER]

CDEGS  
4 [ENTER] [ENTER] [ENTER]

MNPS [SMP3].

```

4:      'X'
3:      'X^2'
2:      'X*Y'
1:      'X*Y^2'
FABST CDEGS MNPS SMP3 F M

```

```

3:      4
2:      4
1:      4
FABST CDEGS MNPS SMP3 F M

```

```

1:      1.88640696485E-2
FABST CDEGS MNPS SMP3 F M

```

Applications of multiple integrals proceed just as in the examples above. In every case, construct the multiple integral, express it as an iterated integral, renaming the variables if necessary in order to match the programs above, and use the appropriate program to evaluate it.

## Chapter 12. Just for Fun

The HP 48 can be used for many things in addition to calculus problems. Some of them are so much fun that we cannot resist including them here for the interested reader to play with. Some of them are useful at the same time.

### The Time Value of Money

The following program relates the present value, future value, payment, interest rate, and number of payments in "time value of money" problems. It assumes a regular payment schedule with equal payments and a fixed periodic interest rate. It is suitable for figuring such things as amortization and annuities.

The program creates a formula relating the above values and puts it in the Solver, where the user can enter the known values and solve for the remaining one. Upon completion of the exercise, a QUIT button appears on the VAR menu, allowing the user to delete the variables created and leave the VAR menu undisturbed.

Here is the program:

```
« 'S1' CRDIR S1 '(1- EXP(-N*LNP1(I/100)))*PMT*100/I+PV=-  
(FV*EXP(-N*LNP1(I/100)))' STEQ « UPDIR 'S1' PGDIR » 'QUIT'  
STO 30 MENU  
»
```

checksum: # 34542d

Let's illustrate. Suppose you want to buy a car for \$5000. You can get 13.5% annual interest at the credit union for a four-year contract, and you want to find out what the monthly payment will be. Press **TVM**. Then enter the data: the present value is 5000, the future value is 0, and the number of months is 48; the annual interest rate is 13.5, so the monthly interest rate is  $13.5 \div 12$ . Then we want to solve for PMT. So we type

**TVM**

5000 **PV** 0 **FV** 48 **N** 13.5 **ENTER** 12 **÷**  
**I** (left-shift) **PMT**.

```
RAD 1USR  
{ HOME DEMO S1 }  
4:  
3:  
2:  
1:  
N I PMT PV FV EXP/E  
  
RAD 1USR  
{ HOME DEMO S1 }  
4:  
3:  
2:  
1: PMT: -135.381614767  
N I PMT PV FV EXP/E
```

The negative sign indicates money going out.

To exit the Solver, press

VAR

```

RAD                                     IUSR
{ HOME DEMO S1 }
4:
3:
2:
1:
I N FV PV PMT C

```

and then NXT

```

RAD                                     IUSR
{ HOME DEMO S1 }
4:
3:
2:
1:
QUIT EQ

```

and finally QUIT.

```

RAD                                     IUSR
{ HOME DEMO }
4:
3:
2:
1:
ANIM CHMS CHIL MUSE TMM

```

The VAR menu is returned to its original state.

## Chaos

Chaos is a subject that is gaining a lot of interest lately. The simplest approach to chaos is in terms of a non-linear feedback system, a non-linear function whose output is the input for the function again. Let's make it concrete.

Suppose we have a population of creatures with non-overlapping generations, such as temperate-zone insects. Each year's population depends on the previous year's population, as well as on other factors. The function that describes this yearly dependence is called the *reproduction curve* for the species. A typical reproduction curve depends on both the number of individuals from the previous year and the difference between that number and the *carrying capacity* of the environment, the number of individuals the environment will support. We will use the reproduction curve

$$f(p) = ap(1 - bp),$$

where  $a$  and  $b$  are constants and  $p$  is the previous year's population.

If we analyze the function  $f$ , we see that the constant  $b$  determines the maximum size of the population. Because a population cannot be negative, the factor  $1 - bp$  must be positive, so  $p$  must be less than  $\frac{1}{b}$ . That is,  $\frac{1}{b}$  is the carrying capacity of the environment. Also, the

function  $f$  has as its maximum the number  $\frac{a}{4b}$ ; if this number is to be less than  $\frac{1}{b}$ , then the value of  $a$  must be less than 4. Clearly  $a$  must be positive. The actual value of  $a$  depends on the environment and on factors relating to the species under consideration.

If  $p_1$  is the population one year, then the population the next year will be  $p_2 = f(p_1)$ ; the next year's population will then be  $p_3 = f(p_2)$ , and so forth. The numbers  $p_1, p_2, p_3, \dots$  are the numbers of individuals in the successive generations. These numbers are found by iterating the function  $f$ , starting with  $p_1$ .

The next question is this: Given a particular value of  $a$ , what is the long-term behavior of the population? We discover that for some values of  $a$ , the population behavior is very predictable, but for other values, the behavior is inherently unpredictable.

To have the HP 48 help us see this, we will create an environment that makes for easy computing and also graphical display of the numbers. I call the subdirectory CHAOS, whose menu items are

ASTO	BSTO	P1STO	WEB	ARNG	ATTR
CSCAD	PN	F	A	B	P1
A1	A2	EQ	CPAR	CASCD	PPAR
DA					

DA is a storage container used by the program ATTR.

CASCD is a container used for storing a picture by the programs ATTR and CSCAD.

EQ is for storing a function to be graphed, and is used by the program WEB.

A1 and A2 are containers used by ATTR.

A, B, and P1 are containers for storing the values of the constants  $a$  and  $b$  and the first population  $p_1$ .

F is the container holding the function ' $A * X * (1 - B * X)$ '. We switch to the variable  $X$  to make graphing easier.

PN is a program that takes a given population from the stack and tells the next population. It is `« DUP 'X' STO F EVAL 'X' PURGE »`.

CSCAD is a program that displays the *attractor* or *bifurcation cascade* created by ATTR. The program is `« CASCD PICT STO CPAR 'PPAR' STO GRAPH »`.

ATTR is a program for drawing the attractor of the reproduction curve, using the current value of  $b$  and a range of values of  $a$ . The program computes 200 generations of the population and then plots the next 50 generations in the same column of pixels for 131 values of  $a$ . The resulting picture often shows a bifurcation or *period doubling* of the population, quickly degenerating into chaotic behavior. It takes an hour or so to construct the cascade. The program is

```
« ERASE A1 0 R→C PMIN A2 1 R→C PMAX PPAR 'CPAR' STO A1 'A'
STO 1 131
  START .5 'X' STO 1 200
    START F EVAL 'X' STO
    NEXT 1 50
    START A F EVAL DUP 'X' STO R→C PIXON
    NEXT A1 1 R→C PVIEW A DA + 'A' STO
  NEXT 'X' PURGE PICT RCL 'CASCD' STO
»
```

checksum: # 32438d

ARNG is a program for storing the beginning and ending values of  $a$  to be used by ATTR in constructing the cascade. The program is

```
« → B C
  « B C MIN 'A1' STO B C MAX 'A2' STO A2 A1 - 130 / 'DA' STO
  »
»
```

checksum: # 11657d

WEB is a program for displaying graphically the successive generations, starting with  $p_1$ . The graph of  $f$  and the line  $y = x$  are drawn, and a line segment from the point  $(p_1, 0)$  to the point  $(p_1, p_2)$ . Then a line segment is drawn from  $(p_1, p_2)$  to  $(p_2, p_2)$  on the line  $y = x$ , and then a line segment from  $(p_2, p_2)$  to  $(p_2, p_3)$ . This process continues for 50 generations. The resulting picture, called a web diagram, shows only a few lines if the populations are periodic, but many lines if the behavior is chaotic. The similarity of the picture to a spider's web gives us the name of the diagram. The program is

```
« 0 1 XRNG 0 A B 4 * / YRNG ERASE ( # 0d # 0d ) PVIEW F STEQ
DRAW DRAW 'X' STEQ DRAW P1 0 R→C P1 P1 'X' STO F EVAL DUP 'X'
STO R→C DUP 'P' STO LINE 1 50
  START P X X R→C LINE X X R→C X F EVAL DUP 'X' STO R→C DUP
  'P' STO LINE
  NEXT ( X P ) PURGE GRAPH
»
```

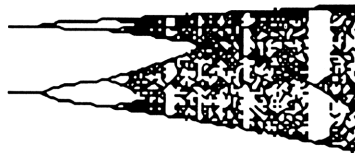
checksum: # 18576d

P1STO is for storing  $p_1$ , and is « 'P1' STO ».

BSTO is for storing  $b$ , and is « 'B' STO ».

ASTO is for storing  $a$ , and is « 'A' STO ».

By playing with the web diagram and by looking at the attractor, you can discover that for  $0 < a < 1$  the population dies out and for  $1 < a < 3$  the population stabilizes. For  $3 < a < 3.5$  the population becomes periodic of period 2. Somewhere in the vicinity of  $a = 3.5$  the population's period doubles to 4, and for a slightly larger value of  $a$ , jumps to period 8. For only a slightly larger value, the population appears to be chaotic. Here is the bifurcation cascade for values of  $a$  between 3.4 and 3.9:



## Public Key Cryptography

*Public key cryptography* refers to a system of encoding messages so that a message may be transmitted over public channels to a receiver, but only the intended receiver can read it. This scheme was devised by Rivest, Shamir, and Adelman in 1978, and also allows for insuring that only the purported sender could have sent the message. This system makes for convenient and secure communications.

Each participant in a public key system publishes a *public key* consisting of two positive integers, a *modulus*  $M$  and an *encryption key*  $E$ . The participant keeps secret a third positive integer, the *decryption key*  $D$ . Anyone who wishes to send the participant a message uses the encryption key  $E$  and the modulus  $M$  to encode the message as described below, but without knowledge of the decryption key  $D$ , no one (not even the sender) can recover the message from the code.

To encode a message, two steps are taken. First, the message is separated into blocks of letters of a specified length, called a *message block*. Each letter in the message block is replaced by the two-digit integer indicating its usual place in the alphabet: A becomes 01, B becomes 02, ... , Z becomes 26. A space is assigned the number 00. Thus each message block becomes a block of digits twice as long, called a *plaintext block*.

Second, a plaintext block  $P$  is transformed into a *code block* using the so-called *exponential cipher*  $P \rightarrow P^E \pmod{M}$ . That is, the number  $P$  is raised to the power  $E$  and  $P^E$  is divided by  $M$ ; the remainder becomes the code block. The only practical way to carry out the encoding, of course, is by computer.

To decode a communication, the receiver uses the same exponential cipher on each code block  $C$ , but employs his decryption key:  $C \rightarrow C^D \pmod{M}$ . If there are no flaws in the design, then the original plaintext block is recovered.

As you may suspect, the numbers  $M$ ,  $E$ , and  $D$  must be chosen carefully to make this work. Two prime numbers  $p$  and  $q$  are selected and  $M = pq$ . Then  $D$  and  $E$  are chosen so that  $DE \equiv 1 \pmod{(p-1)(q-1)}$ . That is,  $DE$  is one more than a multiple of  $(p-1)(q-1)$ . If this is done, then

$$P \rightarrow P^E \rightarrow (P^E)^D \equiv P^{1+k(p-1)(q-1)} \equiv P \cdot (P^{(p-1)(q-1)})^k \equiv P \pmod{M}$$

because of a theorem of Fermat:  $P^{(p-1)(q-1)} \equiv 1 \pmod{pq}$ .

The encryption can be made secure by choosing large primes  $p$  and  $q$ . It has been shown that the simplest way to "break" the code is to factor the modulus  $M$  to find the primes  $p$  and  $q$ , and then knowing  $E$ , to construct  $D$ . If  $p$  and  $q$  have about 100 digits each, it will take so long to factor  $M$  (it would take years, using the best current technology) that breaking the code will have no practical effect.

A simple public key system that can be demonstrated on the HP 48 is given here. It is designed for message blocks of three letters, so that plaintext blocks have six digits. A modulus should be six or seven digits, but larger than 262626, the largest block of plaintext. For example, if primes  $p = 521$  and  $q = 523$  are chosen, then  $M = 272483$ . If  $E = 2141$  is chosen, then  $D = 37781$ .

The public key environment consists of utilities for storing the numbers  $M$ ,  $E$ , and  $D$ , a routine for carrying out the exponential cipher, a program for encoding a message, and a program for decoding the code blocks received. These all reside in a directory called PKEY.

The utilities MSTO, ESTO, and DSTO take the respective numbers from the stack and store them under the appropriate label.

```

MSTO:  « 'M' STO »
ESTO:  « 'E' STO »
DSTO:  « 'D' STO »

```

The program ENCOD takes a message from the stack, written as a string using only capital letters and the space, and breaks it into message blocks. Each message block is encoded using M and E, and a list of the code blocks is the result. For example, using the values of M and E above, the message "THIS IS A MESSAGE" is transformed into the list (73825 168834 201767 134462 67143 240016). Here is the program:

```

« → S
  «
    WHILE S SIZE 3 / FP 0 ≠
      REPEAT S " " + 'S' STO
      END ( ) 1 S SIZE 3 /
      FOR K S '3*K-2' EVAL DUP SUB S '3*K-1' EVAL DUP SUB S
      '3*K' EVAL DUP SUB → F G H
      «
        IF F " " ==
          THEN 0
          ELSE F NUM 64 -
          END
        IF G " " ==
          THEN 0
          ELSE G NUM 64 -
          END
        IF H " " ==
          THEN 0
          ELSE H NUM 64 -
          END → N Q R
        « '10000*N+100*Q+R' EVAL E EXPCI +
        »
      »
    NEXT
  »
»

```

checksum: # 112d

The program DECOD takes a list of code blocks from the stack and recovers the message they represent. It uses the number D in the exponential cipher, and then recovers the message from the plaintext. For example, using the values of M and D above, the list

{144651 130040 158560 164226 9157 5052} is transformed into the message "HOPE YOU ARE WELL". Here is the program:

```

« DUP SIZE → C L
  « " " 1 L
    FOR I C I GET D EXPCI → T
      « T 10000 / IP DUP
        IF 0 ==
          THEN DROP " "
          ELSE 64 + CHR
          END + T 100 / IP 100 / FP 100 * DUP
        IF 0 ==
          THEN DROP " "
          ELSE 64 + CHR
          END + T 100 / FP 100 * DUP
        IF 0 ==
          THEN DROP " "
          ELSE 64 + CHR
          END +
      »
    NEXT
  »
»

```

checksum: # 20010d

Finally, the routine EXPCI performs the exponential cipher. It takes a block and the exponent (E or D) from the stack and returns the transformed block. Here is the routine:

```

« → P A
  « A P P A P 1 → K1 B K3 K4 K5 K6
    «
      DO
        WHILE K4 1 >
          REPEAT K4 2 / IP 'K4' STO K6 2 * 'K6' STO K3 K3 *
          M MOD 'K3' STO
          END K1 K6 - 'K1' STO K1 'K4' STO 1 'K6' STO K5 K3
          * M MOD 'K5' STO B 'K3' STO
          UNTIL K1 1 ≤
          END K5
        »
      »
    »
»

```

checksum: # 12700d



Because anyone can send a message to a participant in a public key system, it is desirable to know for sure who has sent the message. For example, a bank would be reluctant to act on a request to alter an account unless it were absolutely sure that the owner of the account made the request. A participant in a public key system can send another participant a *signaturized* message, so that the receiver knows that only the sender could have sent the message; the security of the cipher is preserved, and public communication channels can still be used.

A signature is something that is unique to a person; in a public key system, the unique thing about each participant is his decryption key. Therefore, before sending a code block, the sender may *sign* it by applying the exponential cipher with his decryption key to it. The receiver, knowing that it is a signed block, first applies the exponential cipher with the purported sender's published encryption key, and then decodes the output in the usual way. If a sensible message is recovered, then the receiver knows that only the purported sender could have sent the message, for no one else could have used the sender's secret decryption key.

To outline this process, let the sender have modulus  $m$ , encryption key  $e$ , and decryption key  $d$ , and let the receiver have modulus  $M$ , encryption key  $E$ , and decryption key  $D$ . The action on a block  $P$  of plaintext is as follows:

$$\begin{array}{ll}
 P \rightarrow P^E \pmod{M} & \text{encryption} \\
 P^E \rightarrow (P^E)^d \pmod{m} & \text{signaturizing} \\
 (P^E)^d \rightarrow [(P^E)^d]^e \equiv P^E \pmod{m} & \text{designaturizing} \\
 P^E \rightarrow (P^E)^D \equiv P \pmod{M} & \text{decryption}
 \end{array}$$

One practical problem arises here: Because two moduli are involved, it is possible that the output block from one process may be larger than the other modulus. However, if the moduli are large and nearly the same size, the probability of that happening is very small. If it does happen, it will only be in isolated blocks, and a message of any size can be reconstructed from the context of the recoverable portion.

Here are signaturizing and designaturizing routines for the HP 48. They take a list of code blocks from the stack and use the stored values of the modulus and the appropriate key to perform or undo the signaturization.

**SIG:** « DUP SIZE → L N « 1 N FOR K L K GET D EXPCI NEXT N →LIST » »  
checksum: # 63937d

**DESIG:** « DUP SIZE → L N « 1 N FOR K L K GET E EXPCI NEXT N →LIST » »  
checksum: # 62740d

To enable you to play with a public key system a little, here are a few keys that fit the PKEY environment:

M	E	D
291163	127	169023
295927	31	9511
320347	47	40751
297587	17	17441
283469	137	14429
287879	29	29669

## Music on the HP 48

The BEEP command on the HP 48 can be programmed to create tunes--simple tunes, at least, for only one tone can be created at a time. Here is an environment for creating music on the HP 48. It allows for writing a tune in a simple way, composing it into a list that can be played, and selecting the key and the tempo in which it is played.

A tune is a list of notes and time values. A note is expressed in half-steps from the tonic (first note in the octave, or 'do'), with steps up being positive and steps down being negative. For example, if the tonic is C, then F is expressed as 5 and FS (F sharp) is expressed as 6; A below C is expressed as -3. Time values are relative durations. If a quarter note has time value 1, then a half note has value 2, an eighth note has value .5, and a dotted quarter note has value 1.5. For example, the following tune is just the major scale:

```
{ 0 1 2 1 4 1 5 1 7 1 9 1 11 1 12 1 }
```

Here is another tune you should recognize:

```
{ 0 .25 0 .25 2 1 0 1 5 1 4 2 0 .25 0 .25 2 1 0 1 7 1 5 2 0  
.25 0 .25 12 1 9 1 5 1 4 2 2 10 .25 10 .25 9 1 5 1 7 1 5 2 }
```

Once a tune is written, a composer translates the half-step designations into frequencies and rearranges the list for rapid play. Then a player takes the composed list from the stack and "BEEPs it out".

I keep the environment in its own subdirectory, called MUZC. The full menu of this environment is

COMP	PLAY	TEMP	SET.T	SCALE	TONIC
DURA	NOTES				

I will describe these items in reverse order.

NOTES is a subdirectory for storing the notes of the 12-tone scale. Here is a printout for the even-tempered scale; just store the values in containers with the given names.

<b>NOTES</b>	C 523.3	F 698.5
DIR	CS 554.4	FS 740
A 440	D 587.3	G 784
AS 466	DS 622.3	GS 831
B 494	E 659.3	END

DURA (for "Duration") is used to compute the actual duration of a note from the time value of the note and the tempo specified in TEMP. The simple program is « 60 TEMP / ».

TONIC is the container for the frequency of the tonic, or first note in the octave. Its contents are determined by SET.T.

SCALE is a container in which I store the major scale above. It helps me a bit in writing tunes, to remind me of the numbers of half-steps of the various notes.

SET.T (for "Set Tonic") is a program for setting the tonic, or key, in which a tune will be played. It takes a note (A, B, CS, etc.) from the stack and stores the appropriate frequency in TONIC. The program is « NOTES RCL UPDIR 'TONIC' STO ».

TEMP is a container for the tempo. Store a value of about 220 for a tune based on quarter notes.

PLAY is the player, that takes a playable list from the stack and plays the tune. Here is the program:

```
« OBJ→ → D « 1 D 2 / START TONIC * SWAP DURA * BEEP NEXT » »
checksum: # 32213d
```

COMP is the composer, that takes a tune from the stack, transforms it into a playable list, and returns the playable list to the stack. Once a tune is written, the playable list may be stored in place of the tune itself. Here is the program:

```
« DUP SIZE → L S
  « S 2 / 1
    FOR K L K 2 * GET L K 2 * 1 - GET → T
      « '2^(T/12)' →NUM
      » -1
    STEP S →LIST
  »
»
checksum: # 26187d
```

To create your own music, first write the tune as a list of notes (half-steps from the tonic) and time values, and store it under a convenient name, say TUNE. Then put the tune on the stack by pressing TUNE, then press COMP, and when it is done, press PLAY. Edit the tune until it is satisfactory, and then store the playable list (the output of COMP) in TUNE. Thereafter, to play the tune, simply press TUNE and then PLAY.

## Index

- ABC 62
- ABS 19
- absolute value 29
  - function 19
- ABSTO 55, 90, 95, 107
- acceleration 97
- amortization 115
- analytic geometry 35
- animation 21
- annuities 115
- antiderivative 51
- antiderivatives 53
- area 52
- ARNG 118
- ASA 27
- ATTR 117
- axes 16
- $\alpha$ STO 58, 118
- back arrow 2
- BEGX 47
- BISCT 44
- bisection method 43
- boundary value problem 57
- BSTO 118
- bug 30
- carrying capacity 116
  - CQ 24
- CDEGS 113
- CDSTO 55, 95, 111
  - EXQ 59
- chaos 116, 117
- checksum 8
  - LNQ 59
- COL? 37
- collect 13
- combinations 34
- combinatorics 34
- command 2
  - graphics 16
- command line 2
- COMP 124
- CON? 37
- COND 34
- condition number 34
- conic sections 79
- Contour maps 101
- coordinates 19
  - Q 8, 33
- cross product 29
- CRVT 98
- cryptography 119
- CSCAD 117
  - SQ 24
  - TQ 24
- cubic spline 64
- curvature 98
- curve in space 88
- curvilinear motion 96
- cylindrical coordinates 98
- D.PTL 38
- D.PT $\pi$  85
- data points 62
- D→Q 10
- DDTAB 49
- De Moivre's theorem 13
- DECOD 120
- decryption key 119
- definite integrals 51
- derivative 40
  - evaluating 40
  - higher-order 41
  - value 40
- DESIG 122
- determinants 28
- difference table 48
- differential equations 54
- DIFTB 48
- DIRDE 104
- directional derivative 104
- display 16
  - saving 18
- distance 35, 85
  - point to line 38
  - point to plane 85
- divided difference table 48
- DNDX 41
- dot product 29
- DRAGR 90, 95
- DRAS 95
- DRAU 95
- DRAV 95
- DRAW 17
  - equations 18
  - procedures 18
- DSTO 120
- DTAB 48
- DUP 3
- DURA 123
- E.OLD 90
- edit 5
- editing environments 6
- ellipsoid 93
- elliptic paraboloid 93
- ENCOD 120
- encryption key 119
- EQ 15, 17
- equation
  - quadratic 13
  - solving 13
  - numerically 14
- equations
  - differential 54
  - linear 28, 31
- EquationWriter 53, 75
- ERACV 56
- ERASC 62
- erase 17, 18
- ERR 69
- error analysis 69
- error formula 70, 71
- error of integration 107
- ESTO 120
- EULER 57
- Euler's method 57
- EULR 57
- EXCO 13, 81
- expand 13
- EXPCI 121
- exponential 28
- exponential cipher 119
- exponential decay 60
- exponential function 59
- exponential growth 60
- expressions
  - evaluating 14
- EXREG 63
- EXRIJ 31
- extremum 43
- FABST 43, 58, 67, 109, 111, 113
- factor 13
- factorials 34
- families of functions 61
- feedback 116
- FGSTO 102
- FIB 74
- FIBL 74
- Fibonacci sequence 74
- fitting to data 62
- FLOOR 19

- FLOW 54, 55  
 Fractional Exponents 20  
 FRCL 61  
 FSTO 45, 61, 108  
 FTAB 47  
 function  
     exponential 59  
     hyperbolic 59  
     logarithmic 59  
     trigonometric 22  
     zero of 43  
 function of two variables  
     graph 92  
 functions  
     families of 61  
     procedures as 20  
 future value 115  
 Gauss-Jordan reduction 31  
 geometric series 75  
 geometry 35  
 GJRED 32, 87  
 GRADI 103  
 graph  
     multiple 19  
     type 79, 83, 84  
 graphics objects 21  
 graphs  
     functions 16  
 greatest integer function 19  
 GUESS 45  
 higher-order derivatives 41  
 HOME 6  
 hyperbolic functions 59  
 hyperboloid, one sheet 93  
 IERR 51, 107  
 IMP.G 101  
 IMPD 41  
 IMPG 79  
 implicit differentiation 41  
 INFL 43  
 INT.D 52  
 INT1 109  
 INT2 110  
 INT3 112  
 INTCV 56  
 integral curve 56  
 intercept 19  
 interest rate 115  
 isolate 13  
 keyboard 1  
 KSTO 110, 112  
 law of cosines 25  
 law of sines 25  
 least squares curve 63  
 left endpoint rule 69  
 length 53  
 LER 68, 69  
 limits 39  
 line 37, 86, 87  
     normal 42  
     tangent 42  
 LL→P 36  
 LNREG 63  
 logarithms 28  
 logistic model 60  
 LOGREG 63  
 LSTO 112  
 M.INT 108  
 Maclaurin series 76  
 manipulation 13  
 matrices 28  
 matrix 30  
     editing 31  
     inverse 30  
 MatrixWriter 30, 31  
 M→Q 33  
 menu 4  
     page 4  
     reordering 6  
 midpoint 35  
 midpoint rule 71  
 MNPS 113  
 MNST 111  
 mode 22  
 modes 4  
 modulus 119  
 MPR 68, 71  
 MSTO 55, 120  
 music 123  
 N.INT 67  
 natural logarithm 59  
 navigation 6, 87  
 NBISC 45  
 NDIF 48  
 net 91  
 NEWT2 102  
 Newton's method 45, 102  
 NFIB 74  
 NGUESS 46  
 NNWT2 103  
 normal line 42  
 NOTES 123  
 NSTO 58, 68, 109  
 numerical integration 51, 108  
 NVAL 47  
 operation 2  
 order 6  
 P.INT 107  
 P.SRF 89, 94  
 P1STO 118  
 P2LN 63  
 P3→ $\pi$  85  
 parameters 61  
 parametric equations 53  
     chain rule 82  
     graphs 83  
 parametric surface 91  
 partial derivatives 101  
 partial sum 75  
 path integral 107  
 payment 115  
 PCHR 82  
 PCR2 82  
 permutations 34  
 PFACT 11  
 PICT 18  
 PINT 107  
 PIVOT 32  
 pivoting 31  
 PKEY 119  
 plane 85, 87, 91  
 PLAY 124  
 plot parameters 16  
 PLTC 56  
 PN 117  
 POINT 62  
 point of inflection 43  
 polar coordinates 83  
 POLYFIT 64  
 POLYREG 64  
 population 116  
 PPAR 16  
 PP→L 35  
 present value 115  
 prime factorization 10  
 principal value range 24  
 probabilities 34  
 program  
     entering 8  
 PS→L 86  
 public key 119  
 purge 5, 16  
 PWREG 64  
 $\pi$  22  
 $\pi\pi\rightarrow L$  87  
 quadric surfaces 92  
 RAD 4  
 radian mode 22  
 ratio table 48  
 RCLGR 90, 94  
 RCLS 94

- RCLU 94
- RCLV 94
- recall 5
- recursive sequences 74
- reduced-echelon form 31
- regression 63
- renaming 5
- reorder 72
- reproduction curve 116
- RER 68, 69
- resolution number 16
- reverse Polish 2
- Riemann sum rule 68
- right endpoint rule 69
- RND 7, 31
- root 14
- rotation 80
- rotation equations 81
- ROTCO 81
- round-off error 31, 34
- row operations 31
- row-reduce 32
- RSTO 107
- RTAB 49
- Runge-Kutta 56
- SAS 27
- SAVE 62
- scalar 30
- scalar product 29
- SCALE 123
- SCTRP 62
- SDAT 62
- SEQ 73
- sequence 73
- SET.T 124
- shift keys 1
- SIG 122
- SIG.D 7
- SIGN 20
- signature 122
- SIMP 71
- Simpson's rule 71, 108
- slope 43
- slope field 54
- SMP1 109, 110, 112
- SMP2 111, 112
- SMP3 113
- soft key 4, 5
- solution of triangles 25
- Solver 14, 15, 25, 60, 73, 75
- SPCRV 89
- sphere 92
- spherical coordinates 99
- SPLCV 64
- SPLIN 65
- SSA 25
- SSS 26
- SSTO 55, 89
- stack 1
  - entry 2
- stack manipulation 3
- STODT 62
- store 4
- STPSZ 47
- subdirectories 6
- subdirectory 87
- SUM 68
- surface
  - parametric 91
  - of revolution 93
- surfaces 91
- SV 16, 60
- SYM 4
- symbolic integration 54
- SYND 21
- synthetic division 21
- systems of lin. eq. 31
- syzygy 127
- table 46
- tangent line 42
- TANL 42, 43
- TAYC 76
- Taylor polynomials 75, 76
- TEMP 124
- TH.D.G 88
- time value of money 115
- time values 123
- TLIN 42
- TONIC 123
- torus 93
- TRANS 88
- TRAP 71
- trapezoidal rule 71
- triangles 25
- trigonometric functions 22
  - inverse 23
  - values 23
- tune 123
- TWIDL 61
- U.N.V 94
- uncertainty number 51
- UPDIR 7
- User keyboard 1
- var menu 4
- variable
  - isolating 13
- VDIF 97
- vector 29
- vector products 29
- vectors 28
- velocity 97
- VFVAL 104
- visit 5
- volume 52
- VSTO 89
- VVAL 97
- WEB 118
- window 16
- XPRG 40
- XYZST 90, 95
- zero 19, 43
- zoom 39
- zooming 19



















ISBN 0-02-340582-1



9 780023 405822



90000>