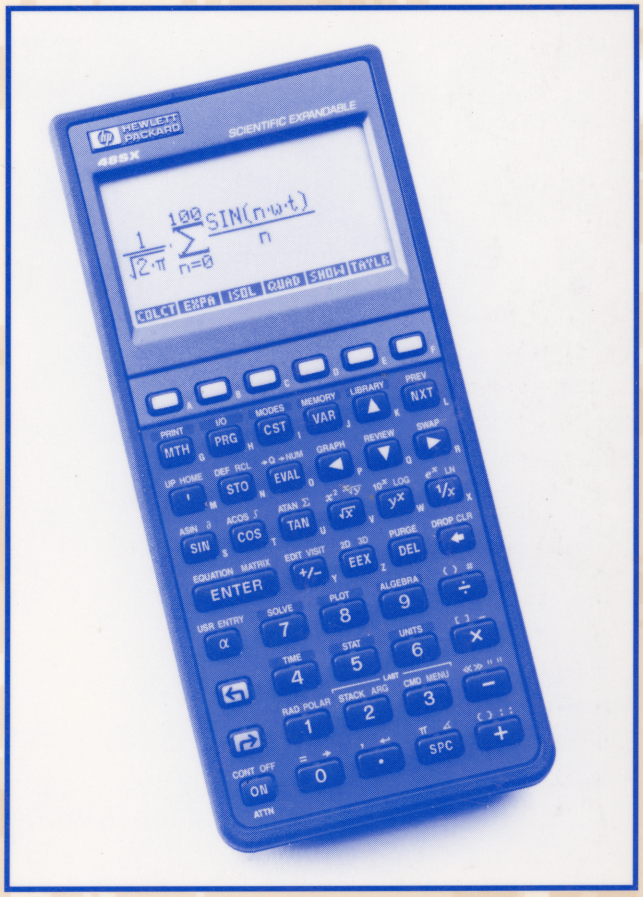
Calculator Enhancement

for a Course in Differential Equations

A Manual of Applications using the HP-48S and HP-28S Calculator



T. G. Proctor, Clemson University

D.R. LaTorre, Clemson University, Consulting Editor

Calculator Enhancement

for Differential Equations

A Manual of Applications using the HP-48S and HP-28S Calculators

T.G. Proctor, Clemson University
D.R. LaTorre, Clemson University, Consulting Editor

Saunders College Publishing
A Harcourt Brace Jovanovich College Publisher
Fort Worth Philadelphia San Diego New York Orlando Austin San Antonio
Toronto Montreal London Sydney Tokyo

Printed in the United States of America.

Proctor: CALCULATOR ENHANCEMENT FOR DIFFERENTIAL EQUATIONS: A MANUAL OF APPLICATIONS USING THE HP-48S and HP-28S CALCULATORS, 1/E

ISBN 0-03-092730-7

234 066 987654321

Contents

Introduction: ... iii
Technology in a Differential Equations Course ... iii
Calculator Preliminaries ... viii
Acknowledgement ... xi
Chapter 1. First Order Initial Value Problems ... 1
Euler and Improved Euler Algorithms ... 2
Graphs of Initial Value Problem Solutions ... 5
Input-Output Functions: Linear First Order Initial Value Problem ... 12
Adaptive Step Size Selection ... 18
Discrete Dynamical Systems ... 23
User Directories: First Order Initial Value Problems ... 26
Chapter 2. Initial Value Problem Variables and Parameters ... 27
Implicitly Defined Solutions of Initial Value Problems ... 27
Use of SOLVE in Application Problems ... 29
Parameter Identification Using Observations on IVP Solutions ... 38
Second Order Linear Constant Coefficient Problems ... 46
On Compartmental Models with a Delay ... 49
Chapter 3. Initial Value Problems: Two Differential Equations ... 56
Euler and Improved Euler Algorithms in Two Dimensions ... 56
Autonomous Problems in Two Dimensions ... 67
Adaptive Step Size Method: Autonomous Differential Equations ... 76
Discrete Systems in the Complex Plane ... 82
Chapter 4. Higher Order Systems ... 85
Solutions for y' = Ay + f(t) ... 85
Euler Algorithm for Vector Equations ... 95
A Nonlinear Problem: the Lorentz Equations ... 97
A Nonlinear Problem: Earth, Moon, Satellite Motion ... 100
Parameter Identification Problems Revisited ... 103
Appendix 1: User Menu Housekeeping ... 106
Appendix 2: User Menu Organization: Directories ... 111
Appendix 3: Starting Points for Newton's Method ... 113
Appendix 4: Runge-Kutta Adaptive Step Size Algorithm ... 117
Appendix 5: Graphs for an Input-Output System ... 121
Indices ... 125-126

Introduction

What advantages does a programmable graphics calculator offer in a differential equations course? How much class time is required for programming? How much previous experience with the calculator is needed for this workbook? We hope to begin answering these questions below.

Technology in a Differential Equations Course

Graphs of important solutions of initial value problems have been a traditional classroom tool for presenting mathematical concepts in a differential equations course. The asymptotic behavior of particular solutions and a comparison between solutions of competing mathematical models are examples. Class time required to produce a graph often impedes progress along a desired line of inquiry. Moreover, in a differential equations course, the lack of an ability to obtain quick, accurate values of definite integrals which require a numerical evaluation or to obtain solutions to matrix equations will deter classroom study of many interesting problems. Thus the use of graphics programmable calculators such as the Hewlett Packard 28S or 48SX adds new vitality to the course. This booklet is to be used as a textbook supplement and addresses the use of an HP-28/48S. Microcomputers can also be employed to stimulate learning especially by the construction of direction fields and the display of multiple solutions of two dimensional dynamical systems. We suggest the calculator be used during class and for homework: microcomputers will usually be restricted to laboratory use.

The calculator programs discussed in these notes do not require significant programming time. The purpose of such technology is to emphasize the concepts presented in the differential equations class and to begin adaptation to the "workstation environment" now present in scientific careers.

Various textbooks can be used with this material. The notes have been developed in a sophomore level course for engineering and science students. Regular homework from this workbook is suggested to encourage a steady pace through the course. The reader will discover that emphasis is given to the construction of graphs of solutions of differential equations to better "understand" the original problem and its solution. The differential equations course supported by this material should include:

an introduction which includes the concept of directions fields, and a presentation of Euler's method and the improved Euler method for solving initial value problems,

sections on first order differential equations with sections on separable and linear problems and equations which may be transformed to such a problem,

sections on mathematical models involving first order equations which includes mixing, population, heating/cooling and falling body problems,

sections on second order differential equations with applications to mechanical spring motion and electric circuits and with some special nonlinear differential equations which may be "solved",

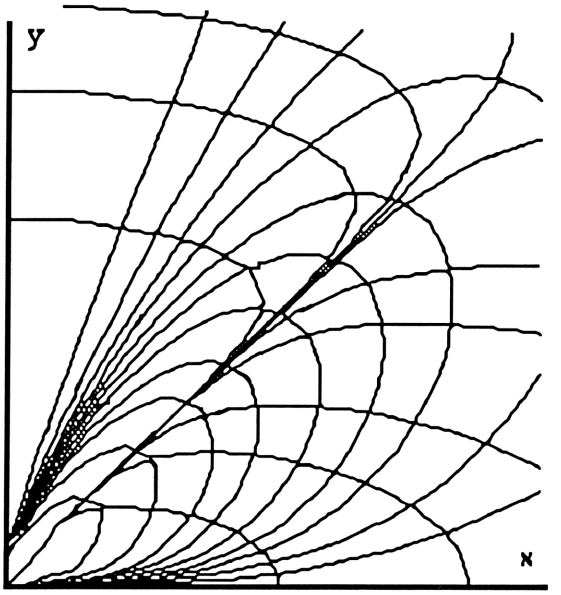
sections on higher order linear differential equations and Laplace transform methods, and

sections on systems of differential equations and matrix methods for linear systems.

The reader should ask several times during the course what advantages does technology gives for understanding concepts and analyzing possible solutions ! The statement that programmable graphic calculators are appropriate classroom tools should also be tested several times during this study ! Here is an example. The solutions of the differential equations

$$\frac{dy}{dx} = \frac{y(3x^2 - y^2)}{x(x^2 - y^2)}, \quad \frac{dv}{dx} = \frac{-x(x^2 - v^2)}{v(3x^2 - v^2)}$$

constitute two families of curves which have the property that members of one family are orthogonal to members of the other at intersection points. The differential equations may be integrated after some effort to give implicit formulas for y(x) and v(x). One of these formulas is quite complicated and the time/effort to construct a graph of the trajectories using this approach is prohibitive. Moreover, this particular aspect of the problem overwhelms the concepts involved. Numerical

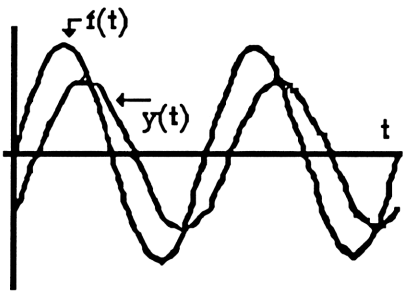


Sample Trajectories of Orthogonal Families of Curves

techniques may be quickly employed to give the figure shown. Now class discussion can proceed to an analysis of the problem; for example, where do the curves have vertical slope, what effects small changes in the differential equations may have, some sense of the sensitivity to changes in initial conditions, etc.

Included in these notes are: (1) calculator programs for initial value problems (Euler method, improved Euler method, an adaptive step size method) with

numerical and graphical output, (2) programs to graph the solutions (outputs) of first or second order linear constant coefficient differential equations for a large set of input forcing functions, (3) programs for solving some nonlinear systems of equations, (4) calculator programs to construct solutions of vector systems y' = Ay + f(t), (5) some suggested problems from application areas such as encountered in population, pursuit, mechanics, and heat models, and (6) some examples of chaotic solutions recently featured in science columns of popular journals. A graph obtained on the HP-28/48S calculator of an input output system dy/dt + y = f(t) in which the periodic output can be readily found by analytical means is shown below. Other types of forcing functions require numerical integration which is easily accomplished on this calculator. Inputs such as triangular waves or square wave as well as complicated combinations of elementary functions can be converted to output solutions quickly by means of our program.



Sinusoidal input, sinusoidal output

It is suggested that graphical emphasis and use of the calculator be made in virtually each class. Ideas present in this material include

- The "problem" dy/dt = f(t,y(t)), y(0) = y_0 defines a function y(t) which satisfies the problem (y can be a vector function) and the solution doesn't change much when the problem is slightly altered.

- Problem comprehension is enhanced by construction of the solution graphs. In particular graphs may draw attention to particular characteristics of solutions such as asymptotic behavior, sensitivity of long range patterns to changes in problem parameters.
- Rapid calculation of solutions of complex equations encountered in falling body, compartmental model, and population problems enhances the study of the original problems.
- The convenience of our computation tools allows the identification of appropriate problem parameters from observation data.
- The study of limiting behavior of interesting discrete and continuous dynamical systems is again made possible by the ease of computation.
- The determination of some eigenvalues/eigenvectors and the quick solution of linear algebraic equations encountered in systems of linear differential equations permits a deeper study of such systems.

Classes using precursors of these notes have been taught for the three years. Student evaluations were collected at the end of each class. Based on the student reactions (verbal and written) approximately 80% of the students felt the calculators were valuable in understanding the material and in working the problems, 10% of the students were unable to recognize benefits from their use: the remaining 10% of the students did not particularly like to use the programmable graphics calculators.

Programs in this workbook have been designed to be understood by the student. Modifications to the programs should be made whenever appropriate. Nevertheless, the programs in this booklet are available on microcomputer disk by the publisher for those instructors who adopt this workbook for classwork. In this way, students who use the HP-48S can read the programs into the calculator and avoid entry errors.

Calculator Preliminaries

Before using this book, the reader should have some familiarity with the HP-28/48S calculator, to the extent of being able to do elementary keyboard calculations, perform routine real number arithmetic, and understand the basics of the stack. To acquire this basic familiarity, simply study Chapters 1-3 of the Owner's Manual.

No further background with the calculator is required in order to begin to explore its capabilities relative to these notes, for we shall develop our skills and understanding as we proceed. Though you should have the Owner's/Reference Manual available to use if needed, our exposition is intended to be self-contained. Readers who wish to acquire an increased level of insight and understanding into the theory and operation of the HP-28/48S are advised to obtain the book "HP-48 Insights"[1] or the book "HP-28 Insights"[2] , by William C. Wickes.

We call your attention to certain features.

- Commonly used keys on the HP-48S include the **MTH, PRG, MODES, MEMORY, VAR, PLOT, GRAPH** menu keys. Some of these keys require a left shift (the orange key) to access. When these keys are activated, a menu appears on the first line of the screen. The white key directly under the desired item will activate that "program feature". The reader should become familiar with the items on each menu. For example on the **MTH** menu are sub-menus labeled **PARTS, PROB, HYP, MATR, VECTR,** and **BASE**. (You will notice a bar over the left side of each of these menu items indicating that each has another level of menus.) Continuing our example under the **MTH PARTS** menu is the second level menu **ABS, SIGN, CONJ,** etc.

---

[1] 1988 by Larken Publications, 4517 NW Queens Avenue, Corvallis, Oregon 97330
[2] 1991 by Larken Publications, 4517 NW Queens Avenue, Corvallis, Oregon 97330

- On the HP-48S calculator to enter capital letters of the alphabet into the command line press the α key, then the desired letter. If several such letters are needed sequentially, pressing the α key twice will leave the calculator in that mode until the α key is pressed again. Warning: when the calculator α mode is active, the menu and some operation keys are inaccessible until the α mode is turned off. Uncapitalized letters are obtained by pressing the α key, then the left shift key, then the desired letter.
- Commonly used keys on the HP-28S include the **MODE, LOGS, PLOT, TRIG, SOLV,** and **USER** menu keys on the right keyboard and the **ARRAY, COMPLEX, STRING, LIST,** and **MEMORY** menu keys on the left keyboard. Some of these keys require a shift ■ to access. When these keys are activated, a menu appears on the first line of the screen. The white key directly under the desired item will activate that "feature". The reader should become familiar with the items on each menu. Examples are **SIN, COS,** etc on the **TRIG** menu, **LN** and **EXP** on the **LOGS** menu, **STEQ, SOLVR** on the **SOLV** menu, **STEQ, RCEQ, PMIN, PMAX,** and **DRAW** on the **PLOT** menu, etc. Other keys used in these notes include **R→C** and **C→R** on the **COMPLEX** menu, **→LCD** and **LCD→ STRING** menu and **→LIST** and **LIST→** on the **LIST** menu.
- Finally, some keys display different characters from what is on them:

Key	Display	Key	Display
[÷]	/	[x^2]	SQ
[1/x]	INV	[$\sqrt{x}$]	$\sqrt{\ }$
[×]	*	[$\frac{d}{dx}$]	∂

Real and complex numbers are two of the eleven different types of "objects" that the HP-28/48S can recognize, manipulate and store. A real number object is the calculator's representation of a 12-digit floating point number:

$$\text{mantissa} \times 10^{\text{exponent}}$$

where the mantissa is a 12-digit number between 1 and 9.99999999999, and $-499 \leq$ exponent ≤ 499. Although the current display mode (**STD**, **FIX**, **SCI** or **ENG**) determines how real number objects are *displayed*, all internal calculations begin by first expanding mantissas to 15 digits and exponents to 5 digits, performing the calculations to that level of accuracy, then rounding back to 12-digit mantissas and 3-digit exponents. All calculations are not accurate to 12 digits: round-off errors from intermediate results may compound as the calculation proceeds.

A complex number object is an ordered pair (x,y) of real numbers, and most arithmetic, logarithmic, exponential and trigonometric operations treat real and complex number objects uniformly. You are free to mix these two object types, and the calculator will return a complex number if any input argument is complex.

You may notice that the 28/48S returns a complex number when asked to calculate an odd-numbered root of a negative real number - like $\sqrt[3]{-1}$ or $\sqrt[5]{-32}$. The result is simply the *principal* root. See the Owner's Manual to obtain the real root.

We shall require no skill or experience in writing programs for the HP-28/48S, but you will need to copy and enter simple programs into your calculator. In doing so, you must be careful and copy the programs exactly as we show them. Special attention should be given to correct spacing because the calculator recognizes commands that are separated by spaces. Instead of spelling out commands from the keyboard, we recommend that you use the menu commands which appear as labels on the various menus; their keystrokes will automatically insert spaces around each

command. To use the menu commands requires some familiarity with their location, but this is readily acquired in the course of entering programs.

Using commands from menu labels will also increase your speed in keying in programs and help avoid errors due to the insertion of extra spaces. For instance, if **R→C** is the desired command, then **R → C** will produce an error message. The desired **R→C** command (rectangular to polar conversion) can be found on the **PRG OBJ** menu on the HP-48S or the **COMPLEX** menu on the HP-28S.

Appendix 1 is a brief review of the procedures for entering, naming, storing, editing, visiting, recalling and purging programs. You may want to consult this appendix initially as you begin to encounter programs.

Acknowledgement

I wish to express my appreciation to The Fund for the Improvement of Postsecondary Education (FIPSE) which has provided funding for this project. Also, I wish to acknowledge the editors and production staff at Harcourt Brace Jovanovich for their commitment to excellence in education through excellence in publishing.

Chapter 1. First Order Initial Value Problems

Programs are given in this chapter for obtaining approximate solutions of first order initial value problems for each of three algorithms. These programs (particularly the first two) require relatively short execution times and can be used in class for many problems. Each of these programs can be used in a graphics program (called **GRAF**) to give pictures of the solutions. The chapter also includes exercises to graph the output solution y(t) of a system dy/dt + ry = f(t), y(0) = 0 for a given input function f(t). Then periodic inputs f(t) are considered and the initial condition is changed so that periodic outputs are obtained. We use the calculator's numerical integration "key" to evaluate an integral in the solution "formula". Finally programs and exercises are included to encourage the students to study the characteristics of solutions obtained in the portion of their course dealing with first order differential equations.

Early in the course students should be aware of direction fields for differential equations dy/dx = f(x,y) and the graphical implications of the uniqueness theorem for solutions of initial value problems Then the student can get a overall view of the solutions and focus on the interpretations. Indeed, solution formulae obtained for initial value problems are only one step in the process of understanding the problems. Sensitivity to parameter values and the asymptotic behavior of solutions may suggest to the modeller how much trust to place in the results.

Euler and Improved Euler Algorithms

The **Euler algorithm** for the solution of an initial value problem results from assuming the slope of the solution of a differential equation $dy/dx = f(x,y)$ is well approximated by the constant $f(x_k, y_k)$ in the interval $x_k \le x \le x_k + h$ and the algorithm is given by $x_{k+1} = x_k + h$, $y_{k+1} = y_k + hf(x_k, y_k)$. (Here y_k is the approximation of $y(x_k)$ and it is assumed that initial values x_0 and y_0 and the step size h are given so the algorithm may be initiated.) Our program is called **EULER** and takes x, y from the stack and returns the results of a single step using Euler's algorithm. It will use the step size, **H**, which is stored and a stored program, **FN**, which takes x,y from the stack and returns f(x,y). "Type" in the program:

<< DUP2 FN H * + SWAP H + SWAP (enter)
'EULER (enter) **STO**

To execute this program, we need a subprogram producing f(x,y) stored in **FN**, a step size stored in **H** and the initial values of x and y on the stack. Try

<< SWAP DROP (enter) **'FN STO** (then) **.1** (enter) **'H STO**
0. (enter) **1.** (enter)

(An alternate form for **FN** is **<< → X Y 'Y' >>**. This form uses local variables and algebraic format.)

Now execute **EULER EULER EULER**, etc. Note: Here we are solving $y' = y$, $y(0) = 1$ using steps H = .1 and get the following results:

x	y	x	y	x	y
.1	1.1	.4	1.46	.7	1.95
.2	1.21	.5	1.61	.8	2.14
.3	1.33	.6	1.77	.9	2.36

and y at x = 1.0 is 2.59, a crude approximation of 2.718....

To aid in recalling quickly exactly the environment and requirements of this program we provide the following formal listing of the program.

Program Name	**EULER**			
Purpose	Generate new values of x and y resulting from one step in Euler algorithm			
Stored Quantities:	**H**	**FN**		
	Input		Output	
	level 2	level 1	level 2	level 1
	x_n	y_n	x_{n+1}	y_{n+1}
	<< DUP2 FN H * + SWAP H + SWAP >>			

Exercise 1.1: To obtain approximate values of the solution of y' = sin(xy), y(0) = 3, execute the following steps: **'FN** (enter) **VISIT** (Use the cursor to change **FN** function to **<<* SIN >>** (alternatively **<<** → **X Y 'SIN(X*Y)' >>**), then (enter). Put initial values 0 3 on the stack and execute **EULER, EULER**, etc. You should get .1 3, then .2 3.03, then .3 3.09, etc. (Make sure the calculator is in **RAD** mode.)

Exercise 1.2: Obtain approximations for the solution of dy/dx = 1 + xsin (2xy), y(0) = 1 at x = .1, .2, .3, .4, .5.

Suppose we want to execute **EULER**, say N, times and observe the output only at x = x_0 + NH. The following program, called **RPT** (for repeat) requires that **N** be stored and initial values of x and y as input and outputs the final values of x and y.

<< 1 N START EULER NEXT>>.

The **Improved Euler algorithm** is another method for approximating the solution of an initial value problem. We give next a short discussion of this algorithm and a program for the algorithm. The method results from assuming the slope of the solution is well approximated by the average of $f(x_k, y_k)$ and a guess at $f(x_{k+1}, y_{k+1})$ in the interval $x_k \le x \le x_k + h$. The algorithm is given by

$$x_{k+1} = x_k + h,\ y_{k+1} = y_k + h[f(x_k, y_k) + f(x_{k+1}, y_k + hf(x_k, y_k))]/2.$$

(Again y_k is the approximation of $y(x_k)$ and x_0, y_0 and h are given so the algorithm may be initiated.) This program is named **IULER** and takes x y from the stack and gives (x+h) (y+h*[f(x,y)+f(x+h,y+h*f(x,y)]/2). Note **EULER** is part of this program.

Program Name	**IULER**			
Purpose	Generate new values of x and y resulting from one step in Improved Euler algorithm			
Stored Quantities:	**H**	**FN**	**EULER**	
	Input		Output	
	level 2	level 1	level 2	level 1
	x_n	y_n	x_{n+1}	y_{n+1}
Instruction	Resulting stack			
<< DUP2 DUP2 EULER	x y x y x+h y+hf(x,y)			
FN 3 ROLLD	x y f(x+h, y+hf(x,y)) x y			
FN + 2 /	x y (f(x+h, y+hf(x,y))+f(x,y)) /2			
H * + SWAP H + SWAP	[y+h*{ f(x+h,y+h*f(x,y))+f(x,y)}/2] x+h			
>>				

Just as in the **EULER** program we require that the program **FN** and the step size **H** be stored before execution. A multiple step program can be obtained by substituting **IULER** for **EULER** in the program **RPT** given just after exercise 1.2.

Try **IULER** using the **FN** program **<< SWAP DROP >>** **H** = .1 and initial data 0 1: execute 9 times. You should get 1 2.7140808--- . (Euler gives about 2.593742--, not nearly so good an approximation of e = 2.71828---.) In general the improved Euler method can be shown to be a better approximation when h is small.

How is an appropriate value of h chosen? If it is decided to use a constant step size throughout the interval of interest $[x_0, x_f]$ one common way to select h is to try a nominal size of h, say $(x_f - x_0)/50$, and calculate the solution approximate y_f at x_f. Then reduce h by half and recalculate the approximate at x_f. If the values agree to your satisfaction (for example, to three decimal places), use the last set of values obtained; if not, reduce h by half and try again.

Exercise 1.3: Try **EULER** on the problem $y' = (y^2 + y)/x$, $y(1) = 1$ with h = .2. Execute 5 times, then reduce h to .1 and execute 10 times. Next execute from the initial value 20 times with **H** = .05. What is being indicated ? Hint: this problem can be solved exactly and has an asymptote at x = 2. Here **FN** could be given by

<< DUP SQ + SWAP / >> or **<< → X Y '(Y^2+Y)/X' >>**

Graphs of Initial Value Problem Solutions

Graphical output can be a powerful tool to study the solutions of initial value problems. Sometimes it is a good idea to graph the values of the approximate solution obtained to check that no crazy things happen. Moreover, we may want a graph of the solution rather than just a set of numbers. Or we may want to compare the solutions of different problems. This is easy on the HP-28/48S by using the

following program, which we will call **GRAF**. This program takes x_0, y_0 from the stack and uses IULER (or **EULER**) to advance **N** steps of size **H**. (**N** is also stored.) On the HP-48S the user should enter the numbers x_{min} x_{max} as **XRNG** and the numbers y_{min} y_{max} as **YRNG** for the graphics screen: on the HP-28S the user must set the lower right corner **PMIN** = (x_{min}, y_{min}) and the upper right corner **PMAX** = (x_{max}, y_{max}), for the graphics screen.

Program Name	**GRAF**	HP-48S version		
Purpose	Graph N values of (x,y) obtained using Euler algorithm			
Stored quantities	**N, H, FN, IULER XRNG YRNG**			
	Input		Output	
	level 2	level 1	level 2	level 1
	x_0	y_0	x_N	y_N
			and graph with cursor	

<< { # 0d # 0d } PVIEW DRAX 1 N START IULER DUP2 R→C PIXON NEXT GRAPH >>

Note: For HP-28S calculator replace the program given above with

<< CLLCD DRAX 1 N START IULER DUP2 R→C PIXEL NEXT DGTIZ LCD→ >>

and store **PMIN, PMAX** instead of **XRNG, YRNG.** Level 1 of the output stack contains the picture as a 'string' which can be reconverted to a graph with the command **→LCD**. Levels 2 and 3 contain x_N and y_N.

GRAF contains a loop in which N new points (x,y) are calculated and plotted. In the HP-28S version there is a command which activates the cursor. **You may want**

to ERASE the graphics screen on the HP-48S. The program **EULER** may be inserted in place of **IULER** so that **GRAF** uses whichever algorithm is desired. Notice also that the last values of x and y remain on the stack after **GRAF** is executed. To restore the stack screen, press **ON**.

We will often be interested in plotting the solutions of several initial value problems on the same graph. On the HP-48S calculator, graphs can be added simply by not erasing the previous result. On the HP-28S it is convenient to have a program for adding a new graph after executing **GRAF** on the first problem. The following program is suggested:

Program Name	**ADGF**	Required for HP-28S only		
Purpose	Graph N values of initial value problem solution (x,y)			
Stored quantities	**N, H, FN, IULER PMIN PMAX**			
Input			Output	
level 3	level 2	level 1	level 2	level 1
previous string	x_0	y_0	x_N	y_N
			and graph with cursor	

<< CLLCD 3 ROLL DUP 'G1' STO →LCD 1 N START IULER DUP2 R→C PIXEL NEXT DGTIZ LCD→ >>

The previous picture is stored in G1 so that the previous graph is not destroyed.

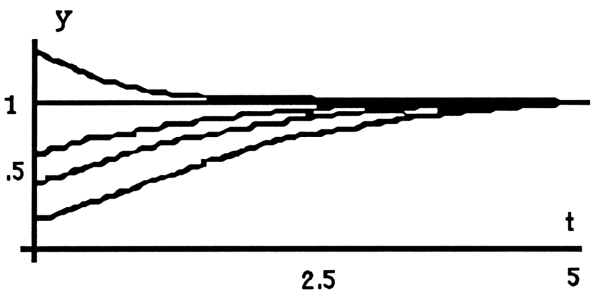
Exercise 1.4: Consider the following differential equation together with several initial conditions and plot the solutions on the same graph.

$$dy/dt = y(1-y),\ y(0) = .2, \text{ then } .4, \text{ then } .6, \text{ then } 1.5$$

where the solutions are plotted for $0 \le t \le 5$ and step size h = .05 is used. Here try

FN: << SWAP DROP DUP SQ - >>.

Enter 0 then .2 on the stack, then execute **GRAF**. (Remember **H** = .05 and **N** = 100 are stored before execution.) Place another initial condition on the stack and add the second solution graph. Notice the solution y = 1 is an attracting solution, i. e. nearby solutions collapse to y = 1 as time increases.

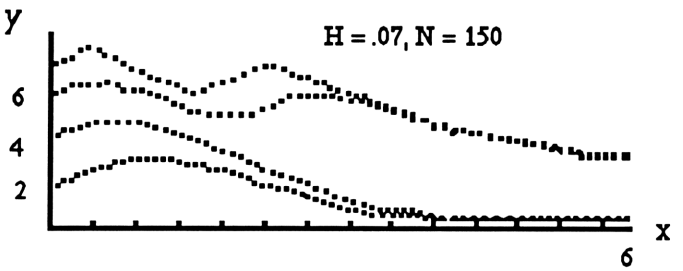


Five solutions of dy/dt = y (1 - y)

Exercise 1.5: Use **GRAF** for $-.5 \le x \le 8$, $-7.5 \le y \le 7.5$ with **N** = 100, **H** = .08 and **FN** given by **<< - SIN >>**. Plot trajectories starting from y(0) = -7.5, y(0) = -1.57, y(0) = 4.71, y(0) = -1.9, y(0) = -2.5, y(0) = 2.5 and y(0) = 4.3.

Exercise 1.6: Try the **GRAF** program on the initial value problem where **FN** is given by **<< * SIN >>** with initial condition y(0) = 3, **H** = .06, **N** = 100 and plot parameters so $0 \le x \le 6$, $0 \le y \le 5$. Select a new starting point y(0) which may be a point chosen by the cursor and **COORD** (on the HP-48S) or **INS** (on HP-28S). Now add the new trajectory after changing (x,y) to x y (on the stack). (If we choose a point x = 0, y(0) = 1.5, get new combination graph, then choose x = 0, y(0) = 1 and get another combination graph, we see the bottom two trajectories approach each other.)

The graphical study of solutions of dy/dx = sin(xy) led to an journal article which gives mathematical proofs for some of the interesting behavior observed in the graphs. See Mills, B. Weisfeiler and A. Krall, "Discovering Theorems with a Computer", *The American Mathematical Monthly*, volume 86 (1979), pages 733-739.



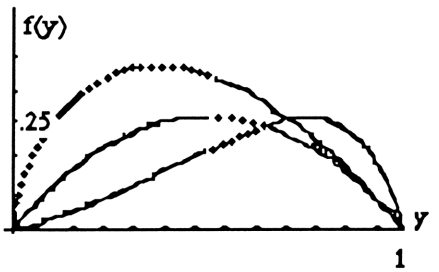
Exercise 1.7: Graph the solutions of the two initial value problems dy/dt = y(1-y), y(0) = .25 and dy/dt = - y ln y, y(0) = .25 (graphic screen parameters $0 \le$ "x" ≤ 5 and $0 \le y \le 1.2$) on the same plot. In this case, we notice the solutions are similar. Which one approaches y = 1 faster ?

Exercise 1.8: Graph the solutions of the two initial value problems dy/dt = y(1-y), y(0) = .25 and $dy/dt = y^2 (1 - y^2)$, y(0) = .25 (graphic screen parameters $0 \le$ "x" ≤ 5 and $0 \le y \le 1.2$) on the same plot. In this case, we also notice the solutions are similar. In which case is a change of concavity apparent ?

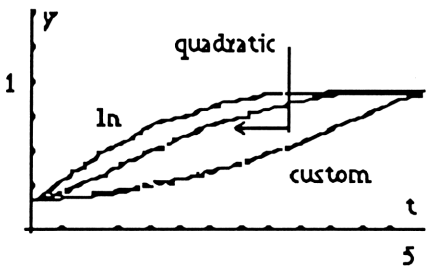
Exercises 1.4, 1.7 and 1.8 give initial value problems which model population growth in a food limited environment. Which model is appropriate? This is a tough question and input from biologists is needed or observation data could be used. Suppose we have experimental data, and we can determine the limiting value of the population and further we can estimate at what fraction of the limiting value of y

an inflection point occurs. In exercise 1.7, the inflection points occur at 36.8% (for the logarithm model) and 50% (for the quadratic model) of the limiting value of y which in both cases is y = 1.

Example: Construct a <u>model</u> dy/dt = f(y) so that if $y_0 < .75$, y(t) has an inflection at y = .75 and $y(t) \to 1$ as $t \to \infty$. Graph the solution starting at y(0) = .2 for $0 \le t \le 5$. Hint: one method is to try f(y) = g(k(y)) where g(y) = y(1-y) and k(y) = $b(e^{ay}-1)$. Notice k(0) = 0 and k'(y) > 0. (k(y) is a nonlinear scaling of the y axis. Since f(1) = 0 we have b = $1/(e^a - 1)$. The requirement f '(.75) = 0 gives b = $.5/[e^{.75a} - 1]$. We use the calculator to solve $e^a + 1 = 2\, e^{.75a}$, (a = 2.4375 and so b = .0957.) The graphs of f(y) vs y and the solutions of y' = f(y), y(0) = .2 are shown.



Right side functions



Dynamics for Example

Exercise 1.9: Set plot parameters to show $-.5 \le x \le 12.56$, $-.5 \le y \le 3$ and put **FN** to **<< → T Y '.5*Y*(EXP(SIN(T))-Y)' >>**. Take **N** = 120 and **H** = 12.56/**N**. Graph the solutions with y(0) = 1, and y(0) = 3. What initial condition gives periodicity ?

Exercise 1.10: Newton's law for a particle falling in a gravity field gives an initial value problem $dv/dt = g - k\, v^r$ where the resistance of the medium is modelled by the term $k\, v^r$. Use **IULER** and **GRAF** to plot the trajectories for the problem for r = 1,

r = 1.5, r = 2, r = 2.5. Plot using **H** = .05, v(0) = .2 for $0 \le t \le 5$ and g = k = 1. This exercise also shows that trajectories from similar models are similar.

Exercise 1.11: Suppose a water storage tank has the shape of the lower half of the ellipsoid $(x^2 + y^2)/a^2 + z^2/b^2 = 1$. Water is introduced into the tank at a net rate of, say $f(t)=.25 \sin [\pi(20-t)/12]$, and evaporation occurs from the exposed surface at a rate k(t) times the exposed surface area, say with $k(t) = (10 - \sin \pi(t-6)/12)/200$. (Here we are assuming the largest evaporation occurs during the daylight hours.) We want the height $\rho(t)$ of water in the tank at time t, given $\rho(0) = \rho_0$. (Note if $\rho > b$ we have water overflow. The volume in the tank is given by

$$V = \pi a^2 \int_0^{\rho} \left[1 - (b - s)^2 / b^2 \right] ds$$

so the balance equation dV/dt = input rate - output rate gives

$$\pi a^2 [\, 1 - (b - \rho)^2 / b^2 \,] \frac{d\rho}{dt} = f(t) - k(t)\, \pi a^2 [\, 1 - (b - \rho)^2 / b^2 \,] .$$

We introduce the scaled variable $u = \rho/b$ to get the equation

$$b \frac{du}{dt} = \frac{f(t)}{\pi a^2 u (2 - u)} - k(t) .$$

Use the following parameters (no physical units intended) and graph the solution for $0 \le t \le 24$ with a step size of 0.08: a = 2, b = 1, and $\rho(0) = u(0) = .8$.

Here is a program for storing values of approximate solutions. We are emphasizing graphics in these notes; however in many cases it is desired to obtain and store the values generated by using **EULER** or **IULER**. The following program computes **N** steps, then records a data point (x, y) in the list **XYV**, and repeats this **M** times. (So x increases from the initial point **MN** steps of length **H**.)

Program Name: **IESV1**

Program to store the **M** values of (x,y) generated by improved Euler algorithm over **N** steps of size **H** in a list **XYV**

Stored Quantities **H FN M N IULER**

Input		Output	
level 2	level 1	level 2	level 1
x_0	y_0	x_{MN}	y_{MN} and list YV

```
<< { } ' XYV' STO DUP2 R→C  XYV + 'XYV' STO 1  M START
  << 1  N  START IULER  NEXT >> EVAL DUP2 R→C XYV +
                'XYV' STO NEXT  >>
```

The list **XYV** can be converted to a graph later.

Input-Output Functions: Linear First Order Initial Value Problem

You can use the calculator algorithm for evaluating definite integrals. Consider the following initial value problem

$$\frac{dy}{dx} = e^{x^2} y^{1+u}\ ,\ y(0) = 1.$$

for $u > 0$. The "separation of variables" technique gives the solution

$$y(x) = \sqrt[u]{\frac{1}{1 - Int}}\ ,\ Int = \int_0^x e^{s^2}\, ds.$$

The program given below takes a number v from the stack and returns the value of the integral of e^{x*x} from 0 to v. The HP-48S version of this program is

```
<< 'X' PURGE  3 FIX  0 SWAP 'EXP(X*X)'  'X'  3 NEG SF ∫  3 NEG CF >>
```

and a corresponding program for the HP-28S is

<< 'X' PURGE X SWAP 0 SWAP 3 →LIST 'EXP(X*X)' SWAP .001 ∫ DROP >>

Notes: (1) →**LIST** is in the **LIST** menu on the left side of the HP-28S calculator. (2) students should note the stack order for integration in a program for their particular calculator: e.g., on the HP-48S calculator, the lower limit is on level 4, the upper limit is on level 3, the integrand is on level 2 and the variable of integration is on level 1. Moreover the -3 flag is set before integration and cleared afterwards.

Values of x and $y^u(x)$ obtained by using the calculator program are

x	Int	$y^u(x)$	x	Int	$y^u(x)$
.5	.545	.455	.75	.918	12.18
.6	.680	.320	.79	.99	103
.7	.833	6.0	.8	1	∞

We note the program suggested above recalculates part of the integral Int for each successive value of x. Clearly we can eliminate this duplication of effort.

Exercise 1.12: Use the integral "key" on the calculator to evaluate the solution at x = .4, .8, 1.2, 1.6 for the problem:

$$\frac{dy}{dx} + y = \sqrt{1 + \cos x}\,,\ y(0) = 1.$$

Suppose a tank which contains V volume units of a mixture of water and a chemical substance receives f(t) units (weight) of the chemical in solution per minute. The chemical and water is vigorously mixed in the tank and the mixture drains from the tank in such a way that constant volume in the tank is maintained. If y(t) is the weight of chemical in the tank at time t, a balance equation gives dy/dt = rate that the chemical enters the tank - rate that the chemical exits from the tank.

This application gives one example of an **important** problem, namely, determine the solution of

$$\frac{dy}{dt} + ry = f(t), \quad y(0) = 0.$$

Here we assume r is a given constant. Commonly the function f(t) is called input to the problem and the solution y(t) is called the output. Other examples of this problem occur in electrical flow problems. The solution of this initial value problem is

$$y(t) = e^{-rt} \int_0^t e^{rs} f(s)\, ds .$$

The input function f is transformed to the output function y = Tf. Notice T(af + bg) = aTf + bTg where a and b are constants and f, g are input functions. This superposition property of the "operation" T shows the transformation T to be a linear operator.

Here we are interested in comparing the graphs of the input functions f to the graphs of output functions y = Tf. Consider the examples,

(a) for input function $f_1(t) = 1$, we have output $y_1(t) = (1 - e^{-rt})/r$ and

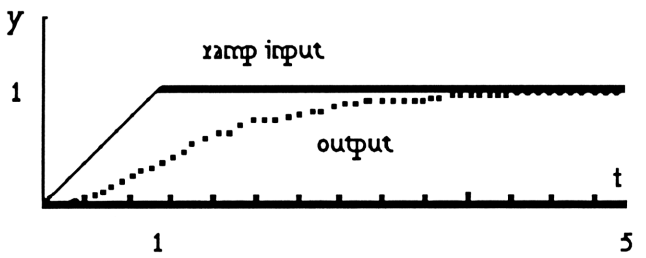
(b) for $f_2(t) = \sin \alpha t$, $y_2(t) = [R \sin(\alpha t - \theta) + \alpha e^{-rt}]/R^2$. Here $R^2 = (\alpha^2 + r^2)$ and $\cos\theta = r/R$, $\sin\theta = \alpha/R$.

There is an obvious similarity between the graphs of the input and output functions. Suppose a signal f(t) = sin t is input into a device which produces output as described above and it is desired to produce a "delayed" version of the signal, say sin $(t - \pi/4)$, after the second term dies out. What value of r will give this delayed

signal? What distortion of another signal, say sin 3t, will be produced by this same device?

If the input signals are not sine/cosine in form, it may be difficult or impossible to find an analytical form of the output; however, a graph of the output may be found by using **GRAF**. (Comment: it is possible to devise a program using the numerical integration capability of the calculator to compute the solution to the precision of the numerical integrator and plot the various y(t) values. However it may be more productive for the student to use the **GRAF** program both to re-enforce familiarity with **GRAF** and to become familiar with the input/output concept with no program distractions.)

For **N** = 100, **H**= 5/**N**, r = 1, the plot parameters set to show $0 \le x \le 5$, $0 \le y \le 1.5$, and the non-periodic input f(t) = Min (t,1) (which is called a ramp function), the output is shown below.



Input-Output System

Exercise 1.13: Let r = 1, **N** = 100, **H** = 3.14/**N** and set the plot parameters so $0 \le x \le 3.14$, $0 \le y \le 1.2$. Use the calculator to draw the following input functions with **DRAW** and the resulting output functions with **GRAF** (or **ADGF** on the HP-28S)

(a) $f(t) = 1 - \sin^4(3t)$, (b) $f(t) = 1 - \sin^{10}(3t)$,

(c) $f(t) = \text{Max}(\sin 6t, 0)$.

The input signals in (a) and (b) are periodic spike-like disturbances of a constant input and the input in (c) is a half-wave rectified sine function.

An unusually observant student may notice that each of the input functions in the previous exercise is periodic with period $T = \pi/3$ and that the resulting output is nearly periodic over the last two periods. This leads one to suspect that the starting condition $y(0) = 0$ is being forgotten and the resulting motion will become periodic. How could we pick an initial condition so that the solution was periodic ? A moment's reflection suggests the condition $y(0) = y(T)$ replace the initial condition $y(0) = 0$. Since the general solution of the differential equation is

$$y(t) = y(0)\, e^{-rt} + y_p(t), \quad y_p(t) = e^{-rt} \int_0^t e^{rs} f(s)\, ds ,$$

this leads to the condition

$$y(0) = \frac{1}{1 - e^{-rT}}\, y_p(T) .$$

Exercise 1.14: For each of the input functions in the previous exercise (where $T = \pi/3$), use the numerical integration capability of the calculator, calculate the appropriate initial condition, and plot the input function using **DRAW** and the output using **GRAF** (or **ADGF**). (We suppose f(x) is stored in **EQ** and $T = 1.047 \approx \pi/3$.) The following programs can be used to calculate y(0):

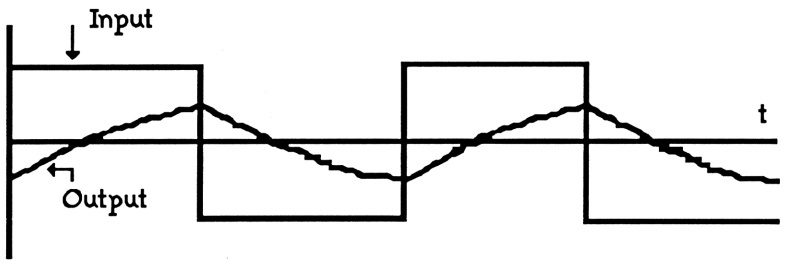
HP-48S version: **<< 2 FIX 'X' PURGE 0 1.047 RCEQ 'EXP(X)' * 'X' 3 NEG SF ∫ 3 NEG CF 1 1.047 EXP SWAP - / >>**

HP-28S version: **<< 'X' PURGE RCEQ 'EXP(X)' * { X 0 1.047 } .01 ∫ DROP 1 1.047 EXP SWAP - / >>**

The function

$$f(t) = 2*CEIL(SIN(t/\pi)) - 1$$

has values given by: for $0 < t < 1$, $f(t) = 2 - 1 = 1$, for $1 < t < 2$, $f(t) = -1$, for $2 < t < 3$, $f(t) = 2 - 1 = 1$, etc. This is called a square wave. The calculator numerical integration "key" and graphing program can handle such a function even though it is not defined at $t = 1$, $t = 2$, etc. The periodic input function and its periodic output are shown for $0 \leq t \leq 4$.



$dy/dt + y = f(t)$

(Notice that the output functions for each of the input functions listed above can be obtained from a table of integrals after several substitutions. An output function for an input function such as $f(t) = 1/(2 - \sin^4(3t))$ could not.)

The student can construct other functions commonly used by engineering studies. For example, the function $u_a(t) = .5[1 + (t-a)/|t-a|] = 0$ when $t < a$ and $= 1$ when $t > a$. This could be called a switch-on function. It can be used in connection with other functions to produce many interesting functions. For example

$$f(t) = \text{Min } (t/.5, 1) - u_{.5}(t)$$

is a ramp function which is switched off at t = .5 . The switch function $u_1(t)$ can be graphed using **DRAW** by storing (**STEQ**) the program

<< IF 'X < 1' THEN '0' ELSE '1' END >> or **'IFTE (X < 1, 0, 1)'.**

Other interesting functions can be obtained using the **MOD** function on the calculator. For example, f(x) = **'MOD(2*X,1)'** (on the HP-28S) or f(x) = **'2*X MOD 1'** (on the HP-48S) will produce repeated ramps of height 1. Finally, functions defined by different formulae in different intervals can be produced by the **IFTE** command: for example, f(x) = 2x for $0 \le x \le .5$, f(x) = 1 - sin (x-.5) for $.5 \le x \le .5 + \pi/2$, x^2 for $x > .5 + \pi/2$ is produced by **'IFTE(X ≤ .5, 2*X, IFTE(X ≤.5+π/2, SIN(X-.5), X^2))'**.

Adaptive Step Size Selection

We will not give much discussion of higher order numerical methods for differential equations such as the Runge-Kutta algorithms. Students are referred to such material in differential equations and numerical analysis textbooks. These algorithms take longer for the HP-28/48S to execute so that classroom exercise selection is somewhat limited. However we do include a program for a **Runge-Kutta** algorithm which calculates at each step an estimate of the local error $|y(t_k) - y_k|$ and determines (adaptively) an appropriate step size in the algorithm for the solution of $y' = f(t, y)$, $y(t_0) = y_0$, $t_0 \le t \le t_f$.

By considering Taylor series for f(t+h, y(t+h)) for the initial value problem

$$dy/dt = f(t, y), \quad y(t_n) = y_n,$$

an algorithm results with increments in t and y given by

$$y_{n+1} = y_n + h[2k_1 + 3k_2 + 4k_3]/9 \quad t_{n+1} = t_n + h,$$
$$k_1 = f(t_n, y_n),\ k_2 = f(t_n+h/2,\ y_n+hk_1/2),\ k_3 = f(t_n+3h/4,\ y_n+3hk_2/4),$$

and the analysis gives the following estimate for the possible error

$$\text{esterr} \leq | 2k_1 + 4k_3 - 6k_2 | /9.$$

See J. Thomas King, *Introduction to Numerical Computation*, 1984, McGraw Hill. **STP1** begins with t_n, y_n on the stack and a stored value of H and continues to reduce H until esterr is less than .01, then stores 2H for the next step and exits with t_{n+1}, y_{n+1} on the stack. (The "tolerance" .01 can be readily modified.)

Program Name	**STP1**				
Purpose	Take one step using Runge-Kutta algorithm with step-size **H** which meets estimated error criterion of .01, selects new trial step **H** for next step				
Stored Quantities	**FN H RK3**				
Input stack	level 2	level 1	Output stack	level 2	level 1
	t_n	y_n		t_{n+1}	y_{n+1}

<< DO RK3 UNTIL .01 < END DELY + SWAP H 2 * + SWAP H 4 * 'H' STO >>

In the subprogram **RK3**, the input (t,y) and the current value of **H** is used to calculate and store a quantity **DELY**, to reproduce (t,y), and to produce an estimate of the error for the current step size. At the end of **RK3** the step size **H** is changed to half of the input size of **H**. When **STP1** finds the error estimate less than .01, the values of t and y are updated (2*H was a successful step) and a new value of **H** is stored, namely 2*successful step size.

Subprogram Name: **RK3**

Purpose: calculate estimated error of algorithm using step size H

Stored Quantities **FN H**

Input		Output		
level 2	level 1	level 3	level 2	level 1
t_n	y_n	t_n	y_n	est error

```
<< DUP2 4 DUPN FN DUP 2 * 6 ROLLD H * 2 / + SWAP H 2 / +
SWAP FN 3 * DUP 4 ROLLD H * 4 / + SWAP H .75 * + SWAP
FN 4 * 3 DUPN + + H * 9 / 'DELY' STO SWAP 2 * - + ABS
9 / H 2 / 'H' STO >>
```

A program listing is given which provides stack status at various program levels.

Command	Resulting stack
DUP2 4 DUPN FN	t y t y t y k_1
DUP 2 *	t y t y t y k_1 $2k_1$
6 ROLLD H * 2 / +	t y $2k_1$ t y t y+Hk_1/2
SWAP H 2 / +	t y $2k_1$ t y y+Hk_1/2 t+H/2
SWAP FN 3 * DUP	t y $2k_1$ t y $3k_2$ $3k_2$
4 ROLLD H * 4 / +	t y $2k_1$ $3k_2$ t y+3Hk_2/4
SWAP H .75 * +	t y $2k_1$ $3k_2$ y+3Hk_2/4 t+.75H
SWAP FN 4 *	t y $2k_1$ $3k_2$ $4k_3$
3 DUPN + +	t y $2k_1$ $3k_2$ $4k_3$ $2k_1$+$3k_2$+ $4k_3$
H * 9 /	t y $2k_1$ $3k_2$ $4k_3$ H[$2k_1$+$3k_2$+ $4k_3$]/9
'DELY' STO SWAP	t y $2k_1$ $4k_3$ $3k_2$
2 * - + ABS	t y |$2k_1$+$4k_3$- $6k_2$|
9 / H 2 / 'H' STO	t y |$2k_1$+$4k_3$- $6k_2$|

As a general rule, we will find that the algorithm will select small steps when the solution is changing rapidly and will increase the step size when the solution is somewhat flat. The reader will notice there are three function evaluations for each step and that the algorithm gives the first three terms correctly in the Taylor series in h for the model problem dy/dt = y, y(0) = 1.

To check this program we consider the problem dy/dt = y, y(0) = 1. Successive executions of **STP1** starting with h = .05 give answers accurate to three decimal places and as the steps are applied h increases to .1, then to .2, then decreases to .1, then to .05 near t = 2.

A second example for this algorithm is for the problem $dy/dt = (y + y^2)/t$, y(1) = 1 (which has solution y = t/(2 - t)). Beginning with **H** = .05 successive executions give

t	y_{app}	y_{exact}	h	t	y_{app}	y_{exact}	h
1.05	1.1053	1.1053	.05	1.375	2.1998	2.2	.025
1.1	1.222	1.222	.05	1.4	2.3332	2.3333	.025
1.15	1.3529	1.3529	.05	1.425	2.4781	2.4783	.025
1.2	1.4999	1.5	.05	1.4375	2.5554	2.5556	.0125
1.225	1.5806	1.5806	.025	1.45	2.6361	2.6364	.0125
1.25	1.6666	1.6667	.025	1.4625	2.7207	2.7209	.0125
1.275	1.7585	1.7586	.025	1.475	2.8093	2.8095	.0125
1.3	1.8570	1.8571	.025	1.4875	2.9022	2.9024	.0125
1.325	1.9628	1.9630	.025	1.5	2.9997	3.0	.0125
1.35	2.0768	2.0769	.025	1.5125	3.1023	3.1026	.0125

The step is reduced again at t= 1.6 to .00625, etc. This algorithm is selecting smaller and smaller h as the asymptote t = 2 is approached. It may be desirable to place a lower bound on the size of h and have an error message when the esterr $\leq$ .01 cannot be maintained.

Exercise 1.15: Use **STP1** for the differential equation $dy/dx = x^6 * y$, $y(0) = .5$ with an initial step **H** = .05. Note the value of H after each execution of **STP1**. Here the step size is not decreased while $0 \le x \le .75$ since x^6 is small and the size of dy/dx is small.

Suppose we are given an interval $0 \le t \le t_{final}$ and an initial value y(0) and wish to apply **STP1** until we reach the time t_{final}. Consider the following program which begins with t_0 y_0 t_{final} on the stack and **H** (an initial stepsize), **FN** stored. The output is a list **YV** which can be readily used to construct a graph.

Program Name	**ARK.1**		
Purpose	Calculate and store values of the solution of an **IVP** using **STP1** from t_0 to t_f		
Stored Quantities	**H FN STP1 RK3**		
	Input		Output
level 3	level 2	level 1	
t_0	y_0	t_{final}	stored list YV

```
<< →TF << { } 'YV' STO DO STP1 DUP2 R→C YV + 'YV'
 STO 2 PICK H + UNTIL TF > 2 PICK TF SWAP - 'H' STO
RK3 DROP DELY + SWAP 2 H * + SWAP R→C YV + 'YV'
                    STO >> >>
```

A variant of this program could be used to solve the differential equation $dy/dt = f(t,y)$ where the "formula" for f(t,y) changes according as a criterion $g(t,y) \le 0$ is or is not satisfied. Note also that **GRAF** can be used directly with **STP1**; however since the step size is not set, a graph with show only the results of **N** steps.

Discrete Dynamical Systems

In some situations, the problem of interest is to determine information concerning the asymptotic behavior of solutions. This occurs often in differential equations; however we will illustrate this type of problem by introducing a new type of problem.

The sequence $\{ y_n \}_0^\infty$ where y_n is given by a recursive function of the form $y_{n+1} = F(y_n)$ is called a discrete dynamical system. Euler's method, the improved Euler method, and Newton's method for finding roots give such systems. Concepts such as constant "solutions", attractive or repelling solutions taken from differential equations are also present in the study of such systems. Discrete dynamical systems (in one dimension) have solutions with more complicated structure than do differential equations. For example, y_n = a constant (for all n) is called a period one solution, $y_{\text{odd } n} = \alpha$ and $y_{\text{even } n} = \beta$ is a period two solution, $y_{3m} = \alpha$, $y_{3m+1} = \beta$, $y_{3m+2} = \chi$ is a period three solution, etc. Consider

$$y_{n+1} = (1 + a)\, y_n - a\, (y_n)^2 \text{ and } y_0 = .1.$$

We want to regard a as a parameter and want to study the effect on the "solution sequence $\{ y_n \}$" as a is varied. In particular we want to study a = 1.8, 2.3, 2.5 and 3 as well as near by values of a.

After several numerical experiments, we find that for a = 1.8 the terms of the sequence approach 1, but for a = 2.3, the terms of the sequence approach 1.18 for even n and .69 when n is odd, etc. This leads us to the following graphical program:

Set the value of a. Calculate the first 50 terms, then graph the values of the next 100, then change the value of a and repeat. To get these cases on a single graph we plot the values of y_n on the horizontal axis and the values of a on the vertical

axis. The following programs can be used: Store **A** = 1.8 and **N** = 100, and set the plot parameters so -.5 ≤ x ≤ 1.5 and 1.6 ≤ y ≤ 3.2, then enter the programs

Program Name	**DDS** HP-48S version
Purpose	Plot the asymptotic value of a discrete system for several values of the parameter a
Stored Quantities	**FN1 DDS1 DDS2 A N XRNG YRNG**
No input:	Output is a graph

```
<< { # 0d # 0d } PVIEW .1 1 12 START DDS1 A .1 + 'A'
STO .1 NEXT GRAPH >>
```

For the HP-28S calculator, replace the program given above with

```
<< CLLCD .1 1 12 START DDS1 A .1 + 'A' STO .1 NEXT
LCD→ >>
```

and store **PMIN** and **PMAX** instead of **XRNG, YRNG**. Subprograms are:

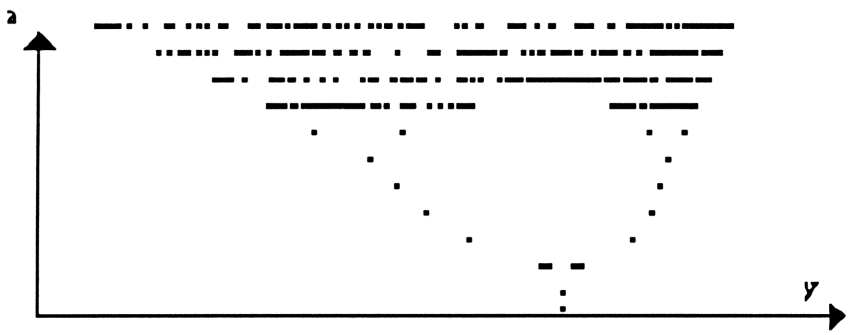
Subprogram Name	**FN1**
Purpose	Create the new value of y, given the previous value

```
<< DUP SQ A * SWAP A 1 + * SWAP - >>
```

Subprogram Name	**DDS1**
Purpose	Execute **FN1** 50 times, call **DDS2**
	<< 1 50 START FN1 NEXT DDS2 >>

Subprogram Name	**DDS2**
Purpose	Repeat **FN1** **N** times, plot points
	<< 1 N START FN1 DUP A R→C PIXON NEXT >>
	Use **PIXEL** for the HP-28S program in place of **PIXON**

(The 50 executions of **FN1** with out graphing allows the sequence to come to "steady state" before the graphing begins.) Execute **DDS**.



Discrete Dynamical System

Exercise 1.16. Change **FN1** to repeat the process for the system

$$y_{n+1} = (1 + a)\, y_n + 2a\, (\cos y_n - 1) \text{ and } y_0 = .1.$$

An appropriate range of the parameter a begins at 2.0.

User Directories: First Order Initial Value Problems

The reader should consider placing the programs presented in this chapter which are likely to be used repeatedly in a directory identified with first order initial value problems. A separate directory can be established for programs for higher order problems. See Appendix 2 for suggestions.

As a final note in this chapter, the HP-48S calculator permits the following program to remind the user for most of the ingredients required for **GRAF**. As written, the user must create and store the FN program which takes X Y from the stack and produces the derivative f(X, Y), then the composite program will prompt for initial conditions, step size, number of steps, etc.

Program Name: **INIT1**
HP-48S Program to set required ingredients for **GRAF**
Stored Quantities: **FN**

```
<< "KEY IN # OF STEPS" " " INPUT OBJ→ 'N' STO "KEY IN
STEP SIZE" " " INPUT OBJ→ 'H' STO "KEY IN XRNG" " "
  INPUT OBJ→ XRNG "KEY IN YRNG" " " INPUT OBJ→
   YRNG "KEY IN INITIAL X" " " INPUT OBJ→ "KEY IN
        INITIAL Y" " " INPUT OBJ→ ERASE >>
```

Chapter 2. Initial Value Problem Variables and Parameters

There are many problems in a differential equations course in which a number (or numbers) satisfying a somewhat complicated equation is needed. We may treat a problem of this type in several ways. In one type of example we may simply use the equation solver routine contained in the calculator. We will give examples of this below. In another type of problem we can use the graphing capability of the calculator to display the inverse of a particular function and thus calculate the graph of a desired solution. A third type of problem, illustrated in the section on parameter identification below, is to solve a vector equation $F(w) = 0$ of a vector variable w. We will give a sequence of programs which may be combined to solve such a problem using Newton's method.

Implicitly Defined Solutions of Initial Value Problems

Implicitly defined solutions may arise arise in the study of first order differential equations, particularly in those problems in which variables are "separated and integrated" or in exact equations.

Example: Graph the solution of

$$dx/dt = 1 - x^{3/2},\ x(0) = 1/2$$

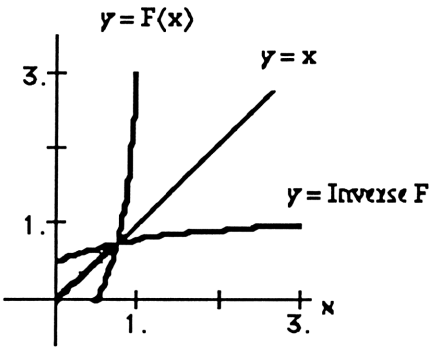
for $0 \le t \le 2.5$. Clearly the solution x(t) will approach 1 as t increases. Using separation of variables, we obtain

$$\int \frac{dx}{1 - x^{3/2}} = t + C.$$

We make the substitution $x = y^2$ and use a partial fraction decomposition for the fraction to obtain the implicit equation $F(x) = t$ where $F(x) = f(x) - f(.5)$ and

$$1.5\, f(x) = \ln\left\{\frac{\sqrt{(1 + x + \sqrt{x})}}{1 - \sqrt{x}}\right\} - \sqrt{3}\ \text{Arctan}\left\{\frac{1 + 2\sqrt{x}}{\sqrt{3}}\right\}.$$

The graph of F(x) versus x for $0 \le x \le .99$ can be obtained quickly on the calculator and is part of the figure shown below.



Since F is an increasing function for $0 \le x \le .99$, there is an inverse function $F^{-1}(w)$ for $F(.5) = 0 \le w \le F(.99) = 2.7$. The solution to our initial value problem is given by $x(t) = F^{-1}(t)$. We include a graphical construction of the inverse function F^{-1} in the figure show above. The following program for the HP-48S, called **INV.F** will overlay $y = F(x)$ with a graph of $F^{-1}(x)$. F should be stored in **EQ**. The input to this program is a pair of numbers A B. The program creates a graph of F^{-1} on the interval $A \le x \le B$.

```
<< DUP2 SWAP - 100 / → A B H << RCEQ CLLCD 'X' STEQ DRAW
STEQ A B FOR I I 'X' STO X EVAL SWAP R→C PIXON H STEP
'X' PURGE GRAPH >>
```

Here, use plotting parameters to show $-.2 \le x \le 2.75$, $-.2 \le y \le 2.75$ with {X .5 .99} as the **INDEP** setting to draw $y = F(x)$. Put .5 .99 on the stack and execute **INV.F**.

For the HP-28S in **INV.F** change the command **PIXON** to **PIXEL** and the command **GRAPH** to →**LCD**. The user should save the picture of F(x) as a string on the stack and combine that string with the output of this program with the command **OR**. To show the composite picture execute →**LCD** (on the **STRING** menu). Warning: the particular graphs F and F^{-1} in this exercise are of limited instructional use on the HP-28S because of the screen dimensions.

Exercise 2.1: Determine the solution of $x' = 1 - x^{2.5}$, $x(0) = .5$ using separation of variables technique. Then use the inverse function to graph x(t).

Use of SOLVE in Application Problems

Suppose we wish a number x so that an equation f(x) = g(x) is satisfied. Enter the equation on the stack enclosed between ' marks. Then **STEQ**. Set plot parameters so that when both sides of the equation are drawn, a crossing is shown. Use the cursor (and the **INS** key on the HP-28S) to locate the approximate crossing coordinates. On the HP-48S execute **ISECT**. On the HP-28S go to the **SOLV** menu and use the **SOLVR** key. Depress the X key, then the shift X key for the result.

Mixing Problem: Initially a large tank holds 2000 gallons of pure water. An stream of 5 gallons per minute with salt content of 2 #/gallon is input into the tank and 4 gallons per minute of the well mixed solution is drained from the tank. When is there Q_0 pounds present in tank ? The usual model dQ/dt = input rate - output rate gives

$$Q = 2\left[2000 + t - \frac{(2000)^5}{(2000 + t)^4}\right].$$

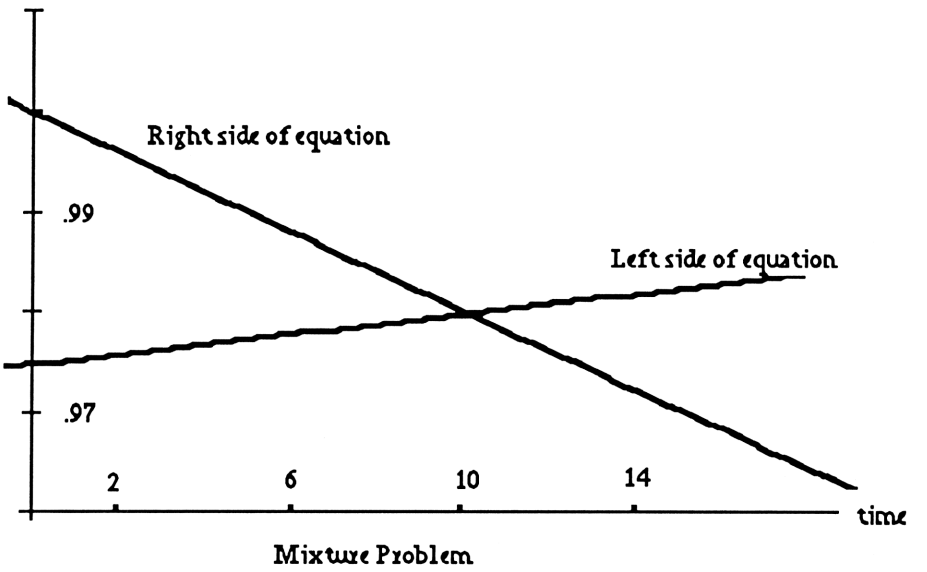
Putting $Q(t) = Q_0$ gives

$$\frac{2000 - \frac{Q_0}{2} + t}{2000} = \left[\frac{2000}{2000 + t}\right]^4$$

to solve for t. For $Q_0 = 100$, we get

$$\frac{1950 + t}{2000} = \left[\frac{2000}{2000 + t}\right]^4;$$

and if we use plotting parameters to show $0 \le x \le 20$, $.9 \le y \le 1$, we get an intersection at about 10 as shown:



Mixture Problem

Exercise 2.2: A tank initially contains 300 gallons of pure water. Brine containing 1.5# of salt per gallon enters the tank at 2 gallons/minute and the well mixed

solution leaves at 3 gallons per minute. When will the tank contain 21 # of salt ? (There may be more than one solution.)

Falling body problem: For mass m = 500 kg, gravity g = 9.81, resistance coefficient k = 50, v(0) = 0, and dv/dt = g - kv/m, the time when displacement = 1000 is

$$1000 = 98.1\left[t - 10\left(1 - e^{-.1\,t}\right)\right]$$

$$1981 - 98.1\,t = 981\,e^{-.1\,t}.$$

If we use plotting parameters to show $0 \le x \le 20$, $0 \le y \le 500$, we obtain a graph with intersection at t = 18.64.

Population Problem: In a logistic population model $dp/dt = ap - b\,p^2$ with parameters a, b and $p_0 = 3.93$, p(50) = 17.07 we get

$$b\,p_0 = \frac{a\left(\frac{3.93}{17.07} - e^{-50\,a}\right)}{1 - e^{-50\,a}}.$$

Now if we take p = 75.99 at t = 110, we get

$$\left[\frac{3.93}{17.07} - e^{-50\,a}\right]\left(1 - e^{-110\,a}\right) = \left[\frac{3.93}{75.99} - e^{-110\,a}\right]\left(1 - e^{-50\,a}\right).$$

Set x = 50 a then 110 a = 2.2x. We wish to solve

$$\left[e^{-x} - .23023\right]\left(1 - e^{-2.2x}\right) = \left[e^{-2.2x} - .05172\right]\left(1 - e^{-x}\right).$$

If we use plotting parameters to show $1 \le x \le 2$, $-0.05 \le y \le .02$, we get a solution at x = 1.53 which means that a = .031. Suppose that data for a year other than t = 50 is available. What would happen if we computed a using that data?

The problem of obtaining a general solution of

$$\frac{d^n x}{dt^n} + a_1 \frac{d^{n-1} x}{dt^{n-1}} + a_2 \frac{d^{n-2} x}{dt^{n-2}} + \ldots + a_n x = 0$$

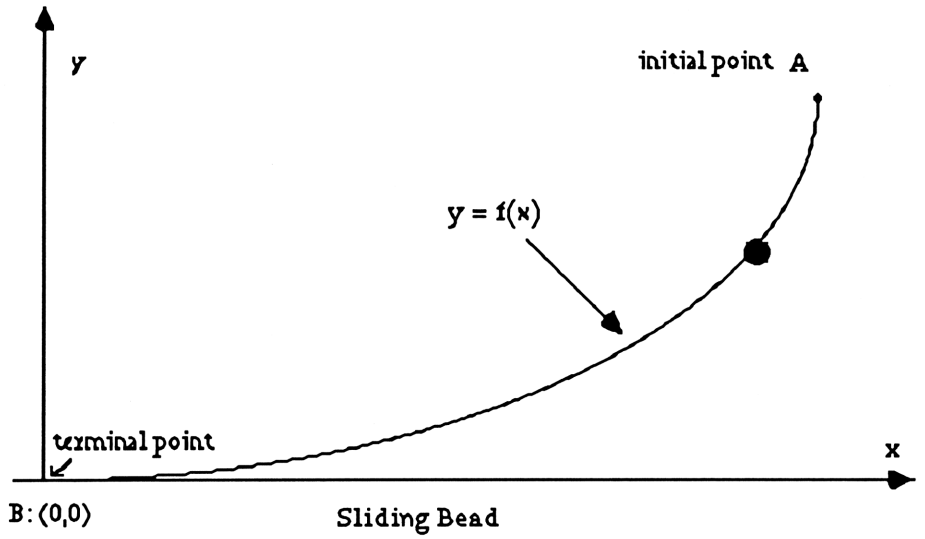
where the coefficients $a_1, a_2, \ldots a_n$ are given constants, is solved by making an "educated guess" for $x = e^{rt}$ for the solution which leads to the algebraic equation $r^n + a_1 r^{n-1} + \ldots + a_n = 0$, sometimes called the characteristic equation. The roots of this equation can be used to construct all solutions of the differential equation. When factors of the left side of this equation are not available a calculator graph of $f(r) = r^n + a_1 r^{n-1} + \ldots + a_n$ gives an approximate solution. **SOLVE** may be used to find a better approximation.

Exercise 2.3: Find all solutions of the form $x = e^{rt}$ to the differential equations

$$\frac{d^3 x}{dt^3} - 3\frac{dx}{dt} - x = 0, \quad \frac{d^4 x}{dt^4} + 2\frac{d^3 x}{dt^3} - 3\frac{d^2 x}{dt^2} - \frac{dx}{dt} + \frac{1}{2} x = 0.$$

Project Exercise: Travel time for a sliding bead as a function of trajectory shape:

We want to specify the shape of a curved wire by a function y=f(x), which connects the point A with coordinates (x_1, y_1) to the origin (0,0) (denoted as point B) so that a bead of mass m will slide along the wire from A to B in a minimum amount of time. The bead begins with initial velocity zero and slides with no friction under the force of gravity g.



The arclength formula,

$$s(x) = \int_{x}^{x_1} \sqrt{1 + \left(\frac{df(r)}{dr}\right)^2}\, dr,$$

the principal of conservation of energy (this is a conservative system),

$$\frac{1}{2} m v^2 + mgy = mgy_1,$$

and the expression for the travel time,

$$T = \int_0^T dt = \int_0^{s(0)} \frac{dt}{ds}\, ds = \int_0^{s(0)} \frac{1}{v}\, ds,$$

leads to the equation

$$T(f) = \int_0^{x_1} \sqrt{\frac{1 + \left(\frac{df(x)}{dx}\right)^2}{2\,g\left(y_1 - f(x)\right)}}\, dx,$$

where we have explicitly noted that T depends on the curve y = f(x). (We have used the technique of changing the variable of integration and the Fundamental Theorem of Calculus.)

We further specialize the example by taking $x_1 = 200$, $y_1 = 100$ and $g = 1$. Later for this case we will find there is a curve such that the travel time T is approximately 25.231. For these parameters the following curves connect the points A and B:

(a) $f(x) = x^2/400$, (b) $f(x) = \exp((x \ln 101)/200) - 1$, (c) $f(x) = 100\,[1 - \cos(\pi x/400)]$.

We will evaluate the travel time integrals given above using numerical integration for each of these curves.

For $f(x) = x^2/400$ the descent time integral becomes

$$T = \int_0^{200} \sqrt{\frac{1 + \frac{x^2}{4\left[10^4\right]}}{2\left[100 - \frac{x^2}{400}\right]}}\, dx.$$

Put x/200 = z, evaluate the integral to get T = 26.779 (attempted accuracy: 0.01). Find T for the functions given in (b) and (c). (Note that so far, no differential equation has arisen.)

The differential equation

$$\frac{dy}{dx} = \sqrt{\frac{k^2 - (y_1 - y)}{y_1 - y}}$$

can be shown to give the minimum time of descent. We have

$$dx = \frac{\sqrt{y_1 - y}}{\sqrt{k^2 - (y_1 - y)}}\, dy.$$

Use the change of variables

$$y_1 - y = k^2 \sin^2 \frac{\varphi}{2}$$

to get

$$dx = -k^2 \sin^2 \frac{\varphi}{2}\, d\varphi, \quad x = C - \frac{k^2}{2}(\varphi - \sin \varphi).$$

At $\varphi = 0$, $x = x_1$ and $y = y_1$ and at $\varphi = \varphi_1$ (to be determined), $x = y = 0$.

Thus the solution to the differential equation along with the transformation y to φ yields a parametric representation $(x(\varphi),y(\varphi))$ of the curve of minimum descent time. There are two constants, k^2 and φ_1, to be determined.

$$x = x_1 - \frac{k^2}{2}[\varphi - \sin \varphi], \quad y = y_1 - \frac{k^2}{2}[1 - \cos \varphi], \quad 0 \le \varphi \le \varphi_1$$

The constraint $x(\varphi_1) = y(\varphi_1) = 0$ leads to the equation

$$k^2 = \frac{y_1}{\sin^2 \frac{\varphi}{2}} = \frac{2y_1}{1 - \cos \varphi_1} = \frac{2x_1}{\varphi_1 - \sin \varphi_1}.$$

We may solve

$$G(\varphi) = \varphi - \sin \varphi - \frac{x_1}{y_1}(1 - \cos \varphi) = 0$$

for $\varphi = \varphi_1$ with the calculator. Notice there is a positive solution. Now determine k^2. Using the differential equation in the expression for the descent time

$$T(y) = \int_0^{x_1} \sqrt{\frac{1 + \left(\frac{dy(x)}{dx}\right)^2}{2g\left(y_1 - y(x)\right)}}\, dx,$$

we obtain the expression

$$T = \int_0^{x_1} \sqrt{\frac{1 + \frac{k^2 - (y_1 - y)}{y_1 - y}}{2(y_1 - y)}}\, dx = \frac{k}{\sqrt{2}} \int_0^{x_1} \frac{dx}{y_1 - y(x)}.$$

But $y_1 - y = k^2 [1 - \cos \varphi]/2$, $dx = - k^2 [1 - \cos \varphi] d\varphi$ so

$$T(y_{opt}) = \sqrt{\frac{k^2}{2g}}\, \varphi_1 .$$

Use the values you get for φ_1 and k^2 to evaluate T. You should find

$$T(y_{opt}) = 25.231.$$

The reader may note that the slope of the optimal curve is infinite at the initial point. This results in a quick start for the sliding bead. The optimal curve is called a cycloid. Optimality is shown in the study of the calculus of variations. Notice however, the exponential curve gives a travel time similar to the optimal curve.

Project Exercise: Travel time up a hill versus initial energy

This project is also concerned with the shape of a unknown function f(x). Suppose we give to a particle with initial position at the origin energy E in the form of initial velocity. The particle sliding on the shape function f(x) (we assume that f is an increasing function) leaves the origin, travels up the "hill" f(x), reaches the limit of its travel at which all its energy has been converted into potential energy and then returns to the origin. We observe the time required to do this (as a function of the initial energy). (This problem is also illustrated by the figure given for the previous project.) The time between the particle's leaving and arriving back at the origin is given by

$$T(E) = \sqrt{2m} \int_0^{x(E)} \frac{\sqrt{1 + [f'(x)]^2}}{\sqrt{E - mgf(x)}}\, dx,$$

and $x(E) = f^{-1}(E/mg)$, that is $E - mgf(x) = 0$.

Suppose the shape of the hill (i. e. the curve f (x)) is one of the functions given in the previous project (i. e. a parabola, an exponential function, or a trignometric function). Graph T vs E for E = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 (numerical integration required).

Now read the first part of the article by Keller on Inverse Problems in the *American Mathematician Monthly*, volume 83, 1976, pages 107-118, and describe what is meant by the inverse problem.

Parameter Identification Using Observations on IVP Solutions

Several mathematical models lead to second order ordinary differential differential equations with constant coefficients. These coefficients are usually obtained from measurements either directly on the physical system or on solutions of the system. Here we will concentrate on a technique of deducing coefficient values using measurements on the solutions.

First an example: A common solution function has the form

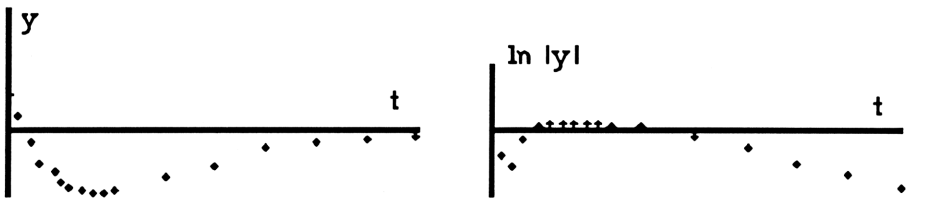
$$y(t) = ae^{-pt} + be^{-qt},$$

where a, b, p, q are parameters. Suppose a set of values { (t,y) } is obtained by making measurements. There is usually some experimental error in measurements so the entire set { (t,y) } will be used to find the parameters. In this example suppose the measurements are:

{ (0, 1), (.1, .30), (.2, -.20), (.3,-.60), (.4, -.88), (.5, -1), (.6, -1.2), (.7, -1.2), (.8, -1.3),

(.9, -1.3), (1, -1.2), (1.5,-1), (2,-.70), (2.5, -.50), (3, -.3), (3.5,, -.17), (4,-.11)}

This list is stored in the calculator as **L1**.



Graphs of Solution Observations

The graphs shown are generated with the HP-48S program

<< CLLCD DRAX L1 LIST→ 1 SWAP START PIXON NEXT GRAPH >>

<< CLLCD DRAX L1 LIST→ 1 SWAP START C→R ABS LN R→C PIXON NEXT GRAPH >>.

Programs for the HP-28S are similar: the command **PIXEL** is used in place of **PIXON** and the command **GRAPH** is omitted.

To get approximations for a, b, p, and q we proceed as follows: suppose $p > q$, then $y(t) = e^{-qt}(ae^{-(p-q)t} + b)$. Since the first term becomes negligible as t increases, $b < 0$ (we replace b with $-|b|$) and a plot of $(t, \ln|y|)$ is a straight line for large t with slope -q. From the second graph we get $q = -.5 \ln(.11/.7) = .925$ from the data points (4, -.11) and (2, -.7). The first data point gives $a = |b| + 1$ (which is, of course, an approximate equation), and $dy/dt(.85) = 0$ gives $p(1+|b|)e^{-.85p} = .42|b|$. Finally we use the approximate equation $\ln -y(t) = \ln|b| - .925t = 0$ at $t = 1.5$ which gives $|b| = 4$, $a = 5$, $pe^{-.85p} = .336$. This equation has two solutions $p = .525$ and $p = 2.2$. Since we want $p > q$, we take $p = 2.2$. These approximate values will be taken later as starting values to an iterative process to determine the parameter values. First however we give a somewhat simpler example.

Suppose { (t, y) } data is given for the solution of an initial value problem and we wish to determine appropriate values of two parameters p and q in a function y = g(t,p,q) to fit the data, say in a least squares sense. That is, we want to choose p and q to minimize the sum

$$\sum_{i=1}^{N} \left[y_i - g(t_i, p, q) \right]^2 .$$

By taking partial derivatives with respect to p and q and setting them to 0 we obtain equations

$$\sum_{i=1}^{N} [\, y_i - g(t_i, p, q)] \frac{\partial g}{\partial p} (t_i, p, q) = 0.$$

$$\sum_{i=1}^{N} [\, y_i - g(t_i, p, q)] \frac{\partial g}{\partial q} (t_i, p, q) = 0.$$

We take the left sides of these equations as components of a vector F, and attempt to solve the vector F(p, q) = 0. We will assume we have starting values for p and q and give an iterative process.

The reader should view this is a special case of the problem of finding a solution of a vector equation F(w) = 0 of a vector variable w. (In this application if we have m parameters, w will be the m vector of parameters and F will be the m vector of partial derivatives.) If the components of the function F are smooth, and we have an approximate solution w_0, then Taylors theorem gives the approximate formula

$$F(w) = F(w_0) + J(w_0)\,(w - w_0)$$

where the matrix J has i, j element $\partial F_i / \partial w_j$. If w is to be a good approximation of the solution, the left side of this equation is zero and we get a "formula" for an improved

vector solution w in terms of the old approximate w_0. We will return to this method in chapter three and use some of the same calculator programs there.

Here is an outline the problem: First we create calculator programs for

$$[\, y - g(t,p,q) \,] \frac{\partial g}{\partial p}(t, p, q) \qquad [\, y - g(t, p, q) \,] \frac{\partial g}{\partial q}(t, p, q)$$

then we will construct a program called **DER** for finding the derivatives of these functions with respect to p, q, etc. After execution, the derivatives can be used to create terms in the Hessian matrix J used in Newton's method.

Next we form the list $\{ (t_1, y_1), (t_2, y_2), \ldots, (t_n, y_n)\}$ by entering the number pairs on the stack, then entering n and the command →**LIST** and store this as **DTA1**.

Finally we create a program called **JACM** to accumulate the data sums in the Hessian matrix after assigning values to p and q. Now we have the ingredients of the Newton formula:

$$\begin{bmatrix} p_{new} \\ q_{new} \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix} - J(p,q)^{-1} F(p,q)$$

and can find new values of p and q.

Many engineering and science problems require the solution of several nonlinear equations. Newton's method is one such algorithm. Most methods to accomplish this can fail under a variety of conditions. Good starting guesses are essential.

HP-28S/48S programs are listed below for Newton's method for this problem. Here we assume there are M parameters and N data points

1. Store the value of M in **M** and M parameters in a list named **PL** = {P Q } : 2 parameters (or in the case of five parameters, say **PL** = { P Q R U V}). Make sure each of the parameter "variables" in **PL** has been purged.

2. Purge the variables T and Y and store the components for the M functions F(T, Y, P, Q) in a list named **FL**. For example,

{ '(Y - 3*EXP(-P*T) + EXP(-Q*T))*3*T*EXP(-P*T)'

'(Y - 3*EXP(-P*T)+ EXP(-Q*T))*T*EXP(-Q*T)'}

would result from trying to fit $y = 3\,e^{-Pt} - e^{-Qt}$ to data.

3. The subprogram **DER** will use the calculator's ability to take appropriate derivatives of the functions in **FL**

Subprogram Name	**DER**
Purpose	Creates list **JL** puts the **FL** functions on the stack and executes **DERA** M times

<< { } 'JL' STO FL LIST→ 1 SWAP START DERA NEXT>>

by calling the subprograms

Subprogram Name	**DERA**
Purpose	Creates M -1 more copies of the first element on the stack for use in the next subprogram

<< 1 M 1 - START DUP NEXT DERB >>

Subprogram Name	**DERB** (replace ∂ in HP-28 program with **d/dx**)
Purpose	Takes M copies of a function in **FL**, creates the derivatives with respect to each parameter in **PL** and stores them in **JL**

```
<<1 M FOR I PL I GET ∂ M 1 + I - ROLLD NEXT
    M →LIST JL + 'JL ' STO>>.
```

4. Store the the N elements of data { (T, Y) } in a list **DTA1.**

5. Now create the programs (which assume values are assigned to P, Q)

Program Name	**JACM**
Purpose and	Create matrix **JMAT**, gets a data point t,y calls the subprogram **JEVP** and does this for each data point

```
<< {M M} 0 CON 'JMAT' STO 1 N FOR I DTA1 I GET C→R
    'Y' STO 'T' STO JEVP NEXT >>
```

Subprogram Name	**JEVP**
Purpose	Evaluates the elements in **JMAT** at the data point and adds it to the value to the previous sum in the **JMAT** element

```
<< JL LIST→ 1 SWAP START →NUM M SQ ROLLD NEXT {M M}
    →ARRY JMAT + 'JMAT' STO >>
```

JMAT will be the Hessian matrix of derivatives.

6. Now for **FVEC** (F vector). In the following P, Q values have been assigned.

Subprogram Name	**FACM**
Purpose	Create **FVEC**, get a data point t, y and call **FEV**: do this for each data point

```
<< {M} 0 CON 'FVEC' STO 1 N FOR I DTA1 I GET C→R
'Y' STO 'T' STO FEVP NEXT>>
```

Subprogram Name	**FEVP**
Purpose	Evaluate the functions in **FL** at t,y, P, Q, ...
and	add the value to the previous value stored in **FVEC**

```
<< FL LIST→ 1 SWAP START →NUM M ROLLD NEXT {M}
→ARRY FVEC + 'FVEC' STO >>
```

Procedure: Store **PL, FL, N, M** and execute **DER** to get **JL** (J list). Put a vector [p,q] with initial values of **P** and **Q** on the stack and execute a program **NST1** given by

```
<< DUP OBJ→ DROP 'Q' STO 'P' STO JACM FACM FVEC JMAT / >>
```

The result is a copy of the old value of [**P,Q**] and the increment [Δ**P**, Δ**Q**]. Execute the command - and repeat. (For the HP-28 program replace **OBJ→** with **ARRY→** .)

Exercise 2.4: Use starting values p = .25, q = 2 and the data to determine p and q in

$$y = 3\,e^{-pt} - 2\,e^{-qt}:$$

{ (0, 1), (.4, 1.89), (.8, 2.01), (1.2, 1.9), (1.6,1.72), (2,1.53),
(2.4, 1.34), (2.8, 1.18), (3.2, 1.03), (3.6, .903), (4, .79), (5, .57) }

Exercise 2.5: Suppose p = .33 and q = 2.5 and the resulting function is a solution of

$$\frac{d^2y}{dt^2} + b\frac{dy}{dt} + ky = 0$$

Determine the damping and spring constants in this linear spring motion.

Project Exercise: Data Fit in a Population Problem. Data showing population numbers { p_i } at times { t_i } in a model $dp/dt = ap - bp^2$ is given. For payoff function

$$P(p_0^*, a, b) = \sum_{i=0}^{n} \left\{ p_i - \frac{ap_0^*}{bp_0^* + (a - bp_0^*)\,e^{-at_i}} \right\}^2$$

we wish to choose p_0^*, a and b to minimize P. Good starting values for the three equations obtained by setting the partial derivatives of P with respect to p_0, a, and b to zero can be obtained by using the data in the year t = 0 (1790) and the years when t = 50 and when t = 100. Use Newton's method to solve this problem.

Year	Population	Year	Population	Year	Population	Year	Population
1790	3.93	1840	17.07	1890	62.95	1940	131.67
1800	5.31	1850	23.19	1900	75.99	1950	151.33
1810	7.24	1860	31.44	1910	91.97	1960	179.32
1820	9.64	1870	39.83	1920	105.71	1970	203.21
1830	12.87	1880	50.16	1930	122.78	1980	226.50

Second Order Linear, Constant Coefficient Problems

We consider here some linear second order differential equations with constant coefficients with and without a forcing term. An important differential equation which models the displacement of a linear spring with linear damping is

$$\frac{d^2y}{dt^2} + 2b\frac{dy}{dt} + \omega^2 y = 0.$$

Case I: Damped oscillatory motion: $\omega^2 > b^2$

For $\omega^2 = 9.25$, $b = .5$, $y(0) = 0$, $y'(0) = 3$, the solution is $y(t) = e^{-.5t} \sin 3t$.

Exercise 2.6: Graph $y(t)$, $0 \le t \le 6.28$ with the plot parameters chosen so $-1 \le y \le 1$. For what values of t does y(t) have a relative maximum ? What is the spacing of the relative maxima ? Comment: To better understand the structure of this important graph, add a plot of $y = \pm e^{-.5t}$ to the graph. (On the HP-48S, this can be done by using the **DRAW** command after **EQ** is modified to '$e^{-.5t} = -e^{-.5t}$' without erasing the graphics screen. On the HP-28S, the **DEL** key should be pressed after the first graph is drawn to record the screen as a string on level one of the stack. After modifying **EQ** to '$e^{-.5t} = -e^{-.5t}$', **STEQ** and **DRAW**. Press **DEL** to record this graph as a string, then return to the stack by pressing **ON**. Combine levels one and two of the stack with **OR** (enter) and bring the composite graph to the screen with →**LCD** which is on the **STRING** menu.) This composite graph, shows a solution envelope.

This solution graph is typical of damped oscillatory motion. If other initial conditions or other parameters b and ω^2 are used(with $\omega^2 > b^2$), the solution graph is similar in form, but scales on the t, y axes are modified. For example, if b and ω^2 remain the same but initial conditions are $y(0) = 3$ and $y'(0) = 10.5$, the solution is

$$\begin{aligned} y(t) &= 5e^{-.5t}[.6\cos 3t + .8\sin 3t] = 5e^{-.5t}\sin(3(t+.2145)) \\ &= 5e^{.5(.2145)}e^{-.5(t+.2145)}\sin(3(t+.2145)) \\ &= 5.566e^{-.5(t+.2145)}\sin(3(t+.2145)). \end{aligned}$$

Before we leave this case, let us note that solutions z_1 (t) and z_2 (t) of the differential equation which satisfy the conditions $z_1(0) = 1$, $z_1'(0) = 0$, $z_2(0) = 0$, $z_2'(0) = 1$ are $z_1(t) = e^{-bt}[\cos \mu t + (b/\mu) \sin \mu t]$,

$$z_2(t) = (1/\mu)\, e^{-bt} \sin \mu t, \text{ with } \mu = (\omega^2 - b^2)^{1/2}.$$

Case II. Overdamped motion: $\omega^2 < b^2$

For this case solutions z_1 (t) and z_2 (t) of the differential equation which satisfy the conditions $z_1(0) = 1$, $z_1'(0) = 0$, $z_2(0) = 0$, $z_2'(0) = 1$ are

$z_1(t) = e^{-bt}[\cosh \beta t + (b/\beta) \sinh \beta t]$, $z_2(t) = (1/\mu)\, e^{-bt} \sinh \beta t$, $\beta = (b^2 - \omega^2)^{1/2}$.

Exercise 2.7. Suppose $\omega^2 = 9$, $y(0) = 0$ and $y'(0) = 3$. For the plotting screen set so $0 \le t \le .5$ and $0 \le y \le .5$, graph the solution of the differential equation for each of the following cases: $b = 2$, $b = 2.7$, $b = 2.99$, $b = 3.01$, $b = 3.4$ and $b = 3.7$ and combine into a single graph to compare the solutions to all of the initial value problems. (In three of the cases, you draw $y = 3\, e^{-bt} \sin((9-b^2)^{1/2} t)/(9-b^2)^{1/2}$ and in three of the cases you draw $y = 3\, e^{-bt} \sinh((b^2 - 9)^{1/2} t)/(b^2 - 9)^{1/2}$.)

Exercise 2.8: Obtain the solution of

$$\frac{d^2y}{dt^2} + \frac{49}{4} y = 3 \cos(5t/2), \quad y(0) = \frac{dy}{dt}(0) = 0$$

by analytical methods, then graph the solution for $0 \le t \le 6.28$ using plotting parameters so $-1 \le y \le 1$. To see an envelope of the solution, generate the graph $y = \pm \sin t/2$ and combine the graphs.

Exercise 2.9: The steady state solution to the forced system

$$\frac{d^2y}{dt^2} + 2b\frac{dy}{dt} + \omega^2 y = K\cos\gamma t$$

is $y(t) = M(\gamma)\sin(\gamma t + \theta)$ where

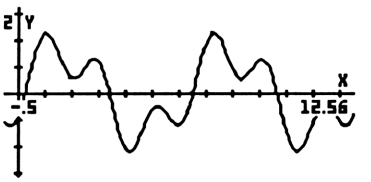
$$M(\gamma) = \frac{K}{\sqrt{(\omega^2 - \gamma^2)^2 + 4b^2\gamma^2}}, \tan\theta = \frac{\omega^2 - \gamma^2}{2b\gamma}.$$

For $\omega = 1$, $K = 1$, and $b = 1$, graph $M(\gamma)$ for $0 \le \gamma \le 2$. (Choose plot parameters so $0 \le M \le 3$.) Repeat for $b = .5$ and $b = .2$. In each case what value of γ gives maximum amplitude ? In each case what is the maximum amplitude ? For $b = .2$, what values of γ give amplitude larger than 2 ?

Exercise 2.10: For $\omega = 1$, $b = .5$, $K = 1$, $L = 1$ and $\gamma = 1$, graph the steady state solution of

$$\frac{d^2y}{dt^2} + 2b\frac{dy}{dt} + \omega^2 y = K\cos\gamma t + L\cos 2\gamma t$$

for $0 \le t \le 12.56$. Use plot parameters to show $-1.5 \le y \le 1.5$.



Steady state solution for
cos t + 5 cos 3t forcing
$\omega = 1, b = .5$

Exercise 2.10 and the figure motivate the study of a differential equation of the same form as that studied above, but with a general forcing (non-homogeneous) term f(t). The method of variation of parameters will give a formula solution which contains an integral. Such a integral can be evaluated numerically on the HP-28/48 when f(t) is given and a graph can be constructed. It is easy to write a program to construct such a graph; however in these notes we will obtain the solution of such a system using a numerical method on the initial value problem. Such a strategy will allow us to concentrate on the initial value problem program rather than on constructing a new program. (However we do give an interesting exercise on this subject using the complex exponential function in Appendix 5.) We will return to this problem in the next chapter.

On Compartment Models with a Delay

Some mathematical models contain assumptions which are known to be only approximate simply because the resulting mathematical problem has an elementary solution. For example, suppose Q(t) is the weight of salt in a tank solution with volume V. Suppose also the tank has an inlet stream of a brine solution with concentration of 2 pounds of salt per gallon entering the tank at 3 gallons per minute,

and the tank drains at 3 gallons per minute. Then dQ/dt = 6 - output rate of salt. The last term in the balance equation for the exiting amount of salt is approximated by 3Q(t)/V. This results from assuming the tank is vigorously mixed so that the concentration of salt in the tank is uniform throughout the tank. The geometry of the tank inlet and drain may be such that a better approximation for this term is 3 Q(t-r)/V. Here the factor Q evaluated at t-r indicates that the salt concentration at the drain is retarded by r minutes from the average concentration Q(t)/V. Of course, the amount of delay may need to be determined by a parameter fit to observed data. In this section we use the graphic and programmable features of the calculator to study such a model.

The problem dQ/dt = 6 - 3Q/500, Q(-50) = 500 has solution

$$Q(t) = 1000\text{-}500\, e^{-.006(t+50)}.$$

The graph of this solution rises smoothly from 500 and approaches 1000 as t increases.

Now consider the modified problem

$$\frac{dQ}{dt}(t) = 6 - \frac{3}{500}\, Q(t\text{-}50)$$

with initial function Q(t) = 500 + 13(t+50)/5 for $-50 \le t \le 0$. (This line approximates the solution of the original model over the interval $-50 \le t \le 0$.) For the interval $0 \le t \le 50$, by direct integration, we obtain $Q(t) = 630 + 3t - 39t^2/5000$, similarly for $50 \le t \le 100$, we obtain $Q(t) = 760.5 + 111(t\text{-}50)/50 - 9*10^{-3}(t\text{-}50)^2 + (117/75)*10^{-5}(t\text{-}50)^3$. An interesting exercise is to graph this composite function and compare the results with the first model.

Exercise 2.11: Graph the function

'**IFTE**(X≤0, 500+2.6*(X+50), **IFTE**(X≤50, 630+3*X-39*X^2/5000,
760.5+2.22*(X-50)-9*(X-50)^2/1000+.117*(X-50)^3/7500))'

for $-50 \le X \le 100$. Use plot parameters to show $500 \le Y \le 1000$. Compare this with the graph of $1000 - 500\, e^{-.006(x+50)}$.

Exercise 2.12: Assume the initial function is $Q(t) = 1000 - 500e^{-.006(t+50)}$ on the interval $-50 \le t \le 0$: derive the solution formulae for the intervals $0 \le t \le 50$ and $50 \le t \le 100$. Plot the result and compare with the function derived above.

The modified model is called a **delay differential equation.** A somewhat less modest example is: $dy/dt(t) = f(t,y(t), y(t-r))$. To uniquely define a solution, the initial function segment , say defined on $-r \le t \le 0$, must be given. Then the problem may be solved sequentially in time steps of length r.

In many cases we want to know the overall solution behavior over a time interval $0 \le t \le M*r$ (here M is a positive integer). Direct extensions of Euler's method or improved Euler's (and other algorithms) method lead can be formulated. We will construct a program for the approximate solution and its graph for

$$\frac{dy}{dt}(t) = a - b\ y(t-r)$$

where the numbers a, b, $r > 0$ are given as well as the initial function y(t) for $-r \le t \le 0$ over an interval $-r \le t \le M*r$. In this program we will take steps of length r/N. Suppose that $y_{-N}, y_{-N+1}, \ldots, y_0, y_1, y_2, \ldots y_k$ approximate the values of the solution at $t = -r$, $t = -r+h, \ldots, t = 0$, $t = r$, $t = 2*r \ldots$ and $t = k*r$. Then

$$y_n = y_{n-1} + \int_{t_{n-1}}^{t_n} \frac{dy}{dt}(t)\, dt = y_{n-1} + .5\ h\ [a - b\ y_{n-1-N} + a - b\ y_{n-N}\ ],\ n = 1, 2, \ldots, N*M$$

results by approximating the integral using the trapezoidal rule. We note that to construct y_n we need the values of y N-1 and N steps back. One way to accomplish this is to construct a list **L1** which initially contains points on the initial function segment and modify **L1** at each step adding the new value y_n and deleting the value y_{n-1-N}. The initial function is stored in the variable **INT.F**, and the output is a graph of the y function, $-r \le t \le M*r$.

Program Name	**DLAY** (for delay system given above)
Purpose	Calculates/plots values of the initial function and the approximate solution for -R ≤t ≤ M*R using step size R/N
Stored Quantities	**R N M INT.F P0 P1 P2 XRNG YRNG**

```
<< { # 0d # 0d } PVIEW INT.F R N / P0 P1 L1 DUP
N 1 + GET 0 SWAP R N / → H << 1 N M * START
H P2 NEXT GRAPH >> >>.
```

We require a subprogram to define the initial "history" of the solution:

Subprogram Name	**P0**
Purpose	Creates a list **L1** containing values on the initial function spaced at R/N time units
Stored Quantities	**R N XRNG YRNG**
Comment: Program creates **L1** and uses the global variable **X** and leaves **H** on the stack.	

```
<< → H << R NEG 'X' STO 1 N 1 + START DUP EVAL
SWAP X H + 'X' STO NEXT DROP N 1 + →LIST 'L1' STO
'X' PURGE H >> >>.
```

and a subprogram to plot the initial function or -r ≤ t ≤ 0:

Subprogram Name	**P1**
Purpose	Plots points created from the list **L1** thus graphing the initial function
Stored Quantities	L1 **R N XRNG YRNG**
Comment: Program creates **L1** and uses the global variable **X** and leaves **H** on the stack.	

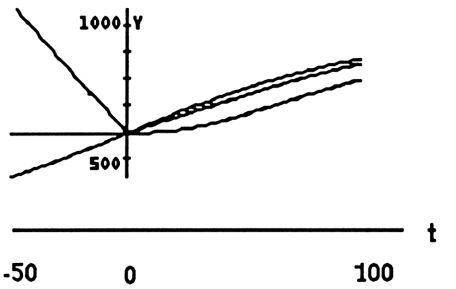
```
<< → H << R NEG 1 L1 1 N 1 + START SWAP GETI
SWAP 3 ROLLD 4 ROLL DUP H + 5 ROLLD SWAP R→C
PIXON NEXT 3 DROPN >> >>.
```

The following subprogram contains the governing differential equation and may be altered to other systems:

Subprogram Name	**P2**
Purpose	Calculates/plots values of the approximate solution of the delay equation for $R/N \le t \le M*R$ using step size R/N
Stored Quantities	**R N XRNG YRNG A B**
Comment: Program **DLAY** leaves a copy of the list **L1** on the stack before executing this subprogram.	

```
<< → H << 3 ROLL DUP DUP 1 GET SWAP 2 GET + B * A
2 * SWAP - H 2 / * 3 ROLL + DUP 3 ROLLD + 2 N 2 +
SUB 3 ROLLD SWAP H + SWAP DUP2 R→C PIXON >> >>.
```

The purpose of the programs given above is to study the responses ($0 \le t \le 100$) for various initial functions. We show below three such responses corresponding to



Delay Model for Mixing Problem
(3 initial functions shown)

initial functions (1) $y(t) = 1000 - 500\, e^{-.006\,(t+50)}$, (2) $y(t) = 629.5$, and (3) $y(t) = 500\, e^{-.006\,(t+50)} + 259.1 - 5.18\, t$ all for $-50 \le t \le 0$. Notice input (1) ≤ input (2) ≤ input (3) for $-50 \le t \le 0$. The outputs have the reverse order: i. e. output (1) ≤ output (2) ≤ output (3). Output (3) = 786 at t = 100, output (1) = 849 at t = 100. The ordinary differential equation model gives the value 797.

Some population biologists have considered population models with a delay term. If a population is large enough that a continuous model gives information and P(t) is the population at time t, then a delay model might take the form

$$\frac{dP}{dt}(t) = r\left[\,1 - \frac{P(t-r)}{K}\,\right] P(t).$$

This model (for r = 0) is called a logistic model for a population size where growth is constrained by a space or food limiting term. The lag term models a delay in the population's perception that space or food resources limits are being approached. The model assumes that such a perception will influence the population to reduce birth rates. An initial function is required on a interval of length r, say $-r \le t \le 0$.

The result is an ordinary differential equation on the intervals $0 \le t \le r$, $r \le t \le 2r$, etc.

Following the ideas introduced for the mixing model, suppose P_n is an approximation for $P(t_n)$ and at each stage of an improved Euler-type algorithm we keep a record of the N+1 values $\{ P_{n-N}, P_{n+1-N}, \ldots, P_n \}$ and where $t_{n+1} - t_n = r/N$. Then by integrating the term $dP(t)/dt$ from t_n to t_{n+1} we get

$$P_{n+1} = P_n + \frac{h}{2} [(1 - P_{n-N}) P_n + (1 - P_{n+1-N})*(P_n + h*(1 - P_{n-N}) P_n)]$$

for $n = 1, 2, \ldots$ We assume that the initial function is given and so the original list $\{ P_{-N}, \ldots, P_0 \}$ is available. The student is encouraged to devise a calculator program for this algorithm and compare the results to those obtained using the ordinary differential equation logistic model.

Chapter 3. Initial Value Problems: Two Differential Equations

This chapter contains a program for obtaining approximate solutions of an initial value problem for each of three algorithms. The initial value problems consists of two differential equations and the initial values of the two dependant variables. The student should note that a second order initial value problem

$$\frac{d^2x}{dt^2} = g(t, x, \frac{dx}{dt}), \quad x(t_0), \quad \frac{dx}{dt}(t_0) \quad \text{given}$$

can be reduced to a first order system of differential equations

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = g(t, x, y)$$

and initial values of x and y. We will show how to graph trajectories of such systems, and study some solution characteristics. A discrete dynamical system in the complex number system (so there are two dependant variables) is given and the asymptotic behavior of solutions is illustrated by graphing points in the Julia set.

Some of the programs given below can be simplified by attacking the problem using vector instead of scaler objects. This approach will be used in the next chapter for the Euler and improved Euler algorithms.

Euler and Improved Euler Algorithms in Two Dimensions

We consider an initial value problem for the equations

$$\frac{dx}{dt} = f(t, x, y), \quad \frac{dy}{dt} = g(t, x, y).$$

The Euler algorithm is based on the same assumption as for one differential equation and is given by $t_{n+1} = t_n + h$, $x_{n+1} = x_n + hf(t_n, x_n, y_n)$, $y_{n+1} = y_n + hg(t_n, x_n, y_n)$. The following program requires we have subprograms **FN** and **GN** which take t x y off the stack and return f(t, x, y) or g(t, x, y). **H** is stored. The program begins with t x y on the stack and terminates with new values of t x y on the stack.

Program Name	**EULE2**					
Purpose	Generate new values of t, x, and y resulting from one step in Euler algorithm					
Stored Quantities	**H FN GN**					
	Input			Output		
	level 3	level 2	level 1	level 3	level 2	level 1
	t_n	x_n	y_n	t_{n+1}	x_{n+1}	y_{n+1}
<< 3 DUPN 3 DUPN GN 4 ROLLD FN 3 ROLLD H * + 3 ROLLD H * + 3 ROLLD SWAP H + 3 ROLLD >>						

The calculator stack contents at various stages of the program is show below:

Command	Resulting stack
<< 3 DUPN 3 DUPN GN	t x y t x y g(t, x, y)
4 ROLLD FN	t x y g(t, x, y) f(t, x, y)
3 ROLLD	t x f(t, x, y) y g(t, x, y)
H * +	t x f(t, x, y) y+H*g(t, x, y)
3 ROLLD	t y+H*g(t,x,y) x f(t,x,y)
H * +	t y+H*g(t,x,y) x+H*f(t,x,y)
3 ROLLD	x+H*f(t,x,y) t y+H*g(t,x,y)
SWAP H +	x+f(t,x,y) y+H*g(t,x,y) t+H
3 ROLLD >>	t+H x+H*f(t,x,y) y+H*g(t,x,y)

Example: For the problem dx/dt = y, dy/dt = -9x, x(0) = 1, y(0) = 0, the solution is x(t) = cos 3t, y(t) = - 3 sin 3t. If we choose **H** = π/48 = .065.. and use **FN, GN** given by **<< 3 ROLLD DROP2 >>** and **<< DROP SWAP DROP 9 * NEG >>** respectively after execution of **EULE2** 4 times we get t = .262, x = .770 and y = -2.26. The correct values of x and y are .707 and -2.12. This somewhat disappointing accuracy can be improved by changing our algorithm for the initial value problem.

The Improved Euler Method for IVP with Two Differential Equations is also based on the same assumptions as in Chapter 1 and the equations for the new values of t, x, and y are given by an obvious extension. Our program for this algorithm deliberately avoids the use of obvious vector subprograms to remain a simple extension of the previous construction.

Program name		**IULE2**				
Purpose		Generate new values of t, x, and y resulting				
from		one step using the Improved Euler Method				
Stored Quantities		**EULER H FN GN**				
	Input				Output	
level 3	level 2	level 1		level 3	level 2	level 1
t	x	y		new t	new x	new y

<< 3 DUPN 6 DUPN EULE2 3 DUPN FN 7 ROLLD GN 8 ROLLD FN + H * 2 / 6 ROLLD GN + H * 2 / + 3 ROLLD + SWAP 3 ROLL H + 3 ROLLD >>

The reader should notice there are three **FN** function and three **GN** function evaluations for each step. The program can be rearranged so there are only two of each per step; however it is not clear that the resulting program is faster for many

problems. The calculator stack contents at various stages of the program is show below. The subscript e means the Euler approximation or the function evaluated at the Euler approximate values.

Command	Resulting stack
<< 3 DUPN 6 DUPN EULE2	t x y t x y t x y t_e x_e y_e
3 DUPN FN 7 ROLLD	t x y t x y f_e t x y t_e x_e y_e
GN 8 ROLLD	t x y g_e t x y f_e t x y
FN + H * 2 /	t x y g_e t x y $[f_e + f]*H/2$
6 ROLLD GN + H * 2 /	t x $[f_e + f]*H/2$ y $[g_e + g]*H/2$
+ 3 ROLLD +	t $y+[g_e + g]*H/2$ $x+ [f_e + f]*H/2$
SWAP 3 ROLL	$x+ [f_e + f]*H/2$ $y+[g_e + g]*H/2$ t
H + 3 ROLLD >>	t + H $x+ [f_e + f]*H/2$ $y+[g_e + g]*H/2$

For the example problem given after the listing for **EULE2**: namely $dx/dt = y$, $dy/dt = -9x$, $x(0) = 1$, $y(0) = 0$ and $H = \pi/48$, after 4 executions of **IULE2**, we get $t = .262$, $x = .704$, $y = -2.13$. The increase in accuracy for this problem leads us (in these notes) to concentrate on the improved Euler algorithm for many problems !

In these notes, we want to emphasize the behavior of solutions of initial value problems over time. A major tool toward such an emphasis will be a graphical presentation of an approximate solution. For the case of two differential equations we may graph x vs t, y vs t or x vs y. A program for any of these cases will be a simple modification of the **GRAF** program of Chapter 1. In that program there was a repeated sequence of steps (a loop) in which we obtained new values of the variables with the improved Euler algorithm, then we created the coordinates of the point we wanted to graph, used the command **PIXON** (or **PIXEL**) and continued to the next step in the loop. We will repeat this scheme for **IULE2**.

Program Name	**GRXY**
Purpose	Graph N values of (x,y) resulting from the improved Euler algorithm which creates a sequence of N values of t, x and y.
Stored Quantities	**N H FN GN EULE2 IULE2 XRNG YRNG**

Input			Output		
level 3	level 2	level 1	level 3	level 2	level 1
t_0	x_0	y_0	t_n	x_n	y_n
			and graph		

<< { # 0d # 0d } PVIEW DRAX 1 N START IULE2 DUP2 R→C PIXON NEXT GRAPH >>

For the HP-28S calculator replace **{ # 0d # 0d } PVIEW** with **CLLCD** and **GRAPH** with **DGTIZ →LCD**.

Program Name	**GRXT**
Purpose	Graph N values of (t,x) resulting from the improved Euler algorithm which creates a sequence of N values of t, x and y.
Stored Quantities	**N H FN GN EULE2 IULE2 XRNG YRNG**

Input			Output		
level 3	level 2	level 1	level 3	level 2	level 1
t_0	x_0	y_0	t_n	x_n	y_n
			and graph		

<< { # 0d # 0d } PVIEW DRAX 1 N START IULE2 3 DUPN DROP R→C PIXON NEXT GRAPH >>

For the HP-28S calculator replace **{ # 0d # 0d } PVIEW** with **CLLCD** and **GRAPH** with **DGTIZ →LCD**.

At this point we return to a problem considered in the last chapter: namely to determine the solution of a non-homogeneous second order differential equation with constant coefficients. The problem is treated in many textbooks (also in the last chapter) for special types of forcing, usually sine or cosine forcing functions. A model for an elastic spring with damping and with external forcing f(t) or a model for a simple electrical circuit loop with external voltage is:

$$\frac{d^2x}{dt^2} + 2b\frac{dx}{dt} + \omega^2 x = f(t),\ x(0) = \frac{dx}{dt}(0) = 0,\ \omega^2 > b^2 .$$

The solution is given by

$$x(t) = \frac{1}{\sqrt{\omega^2 - b^2}} \int_0^t e^{-bt} \sin\sqrt{\omega^2 - b^2}\,(t-s)\, f(s)\, ds .$$

The exercise to be given below is to graph the output function x(t) for a given input function f(t). First some examples: for

$$f(t) = 1:\ x(t) = \frac{1}{\omega^2}\left[1 - e^{-bt}\left(\cos \mu t + \frac{b}{\mu}\sin \mu t\right)\right]$$

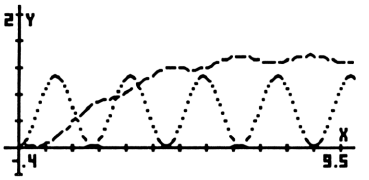
where $\mu^2 = \omega^2 - b^2$ and for

$$f(t) = K\cos \gamma t:\ x(t) = \frac{K}{\sigma}\sin(\gamma t + \theta),\ \sigma = \sqrt{\left(\omega^2 - \gamma^2\right)^2 + 4b^2\gamma^2}$$

where $\tan\theta = (\omega^2 - \gamma^2)/(2b\gamma)$. (Here only the steady state solution is given: the other term in the complete solution is multiplied by e^{-bt}.)

As indicated at the beginning of this chapter, the problem given above is equivalent to the pair of differential equations $dx/dt = y$, $dy/dt = f(t) - 2by - \omega^2 x$.

Example: Take $\omega^2 = .41$, $b = .5$ (so $\mu^2 = .16$) and $f(t) = \sin^2(1.5t)$. Set **FN** and **GN** as **<< 3 ROLLD DROP2 >>** and **<< 3 ROLL 1.5 * SIN 2 ^ 3 ROLLD SWAP .41 * + - >>** respectively, put **N** = 100, **H** = 9.42/**N**, and the plotting parameters to show $0 \le x \le 9.42$, $0 \le y \le 2$. Put 0 0 0 on the stack and execute **GRXT**. Next overlay a graph for the input function. The forcing function (input) and solution (output) resulting from this program are shown below.



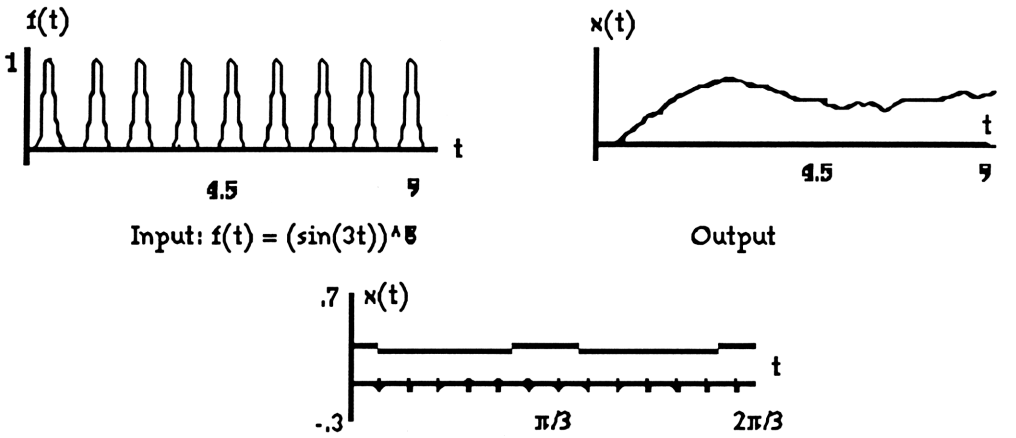
Exercise 3.1 : Find the output graph for $f(t) = 1 - \sin^4(3.14*t)$ for $\mu = 1$, $b = .5$, **N** = 150 and **H** = 6/**N**. Choose plot parameters to show $-.4 \le x \le 6$, $-.4 \le y \le 1.2$. Add the input function graph as an overlay. Comment: The output functions for this input functions can be obtained from a table of integrals after several substitutions using the method of undetermined coefficients and a lot of work. An output function for an input function such as $f(x) = 1/(1 - \sin^4(3.14*x))$ could not.

Suppose the forcing function f(t) is periodic with period length T for the differential equation. If we change the initial conditions so that $x(T) = x(0)$ and $x'(T) = x'(0)$, then the resulting solution is periodic. Moreover if the damping coefficient $b > 0$, then all solutions will eventually be a close approximate of the periodic solution when viewed over one period. We may want to view such a solution without waiting for asymptotic behavior to emerge. Suppose we determine

solution $x_1(t)$ and $x_2(t)$ of the associated homogeneous system so that $x_1(0) = x'_2(0) = 1$ and $x'_1(0) = x_2(0) = 0$, then a general solution is $x(t) = a\,x_1(t) + b\,x_2(t) + x_p(t)$ where $x_p(t)$ is the solution constructed above for 0 initial conditions for x and x'. Expressions for $x_1(t)$ and $x_2(t)$ are given in chapter 2. Use the calculator to compute $x_p(T)$ and $x'_p(T)$ numerically and to solve the periodicity condition for a and b:

$$\begin{bmatrix} 1 - x_1(T) & -x_2(T) \\ -x'_1(T) & 1 - x'_2(T) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_p(T) \\ x'_p(T) \end{bmatrix}.$$

Output for $f(t) = (\sin 3t)^8$, b = .25 and $\mu = 1$ using the initial conditions x(0) = dx/dt(0) = 0 is shown below. This input is periodic with period $\pi/3$. The periodic response is also shown over two periods. Notice the average value for this forcing f(t) is $\pi/6$. Since $f(t) = [f(t) - \pi/6] + \pi/6$, a portion of the periodic response is the constant function with value $\pi/(6\omega^2) = .493$. This seems to be the constant part of the periodic solution as shown.



Exercise 3.2 : Find and graph using the calculator the periodic output response for $f(x) = 1 - \sin^4(3.14x)$ for $\mu = 1$, b = .5, **N** = 100 and **H** = 3.14/**N**. Add input graph.

Pursuit Problem: A rabbit starts at (0,1) and runs along y =1 with speed 1. At the same time a dog starts at (0, 0) and pursues the rabbit with speed 1.3. The dog always proceeds directly toward the rabbit. What is the path of the dog ? This problem can be solved exactly but it takes a while. The equations of motion are:

$$x' = 1.3\,(t-x)/[(t-x)^2+(1-y)^2]^{.5}, \quad y' = 1.3\,(1-y)/[(t-x)^2+(1-y)^2]^{.5}, \; x(0) = 0, \; y(0) = 0.$$

When does capture occur ? A stopping criteria might be $|x - t| < .03$ and $1-y < .01$. In our program we will not give a stopping condition. FN, GN functions for f(t,x,y) and g(t,x,y) are given by:

FN: << 1 - SQ 3 ROLLD - DUP 3 ROLLD SQ + √ / 1.3 * >>
GN: << 1 - DUP 4 ROLLD SQ 3 ROLLD - SQ + √ / 1.3 * NEG >>

We modify the **GRXY** program and call the new program **GR.P** so that the trajectories of both dog and rabbit are shown dynamically. Note effective capture occurs when the objects first close on each other; however after that time the dog effectively turns around, chases backward until the rabbit passes by again, etc. This occurs because the approach is exponential and never really becomes zero. We will return to this problem in the next chapter with a more realistic model.

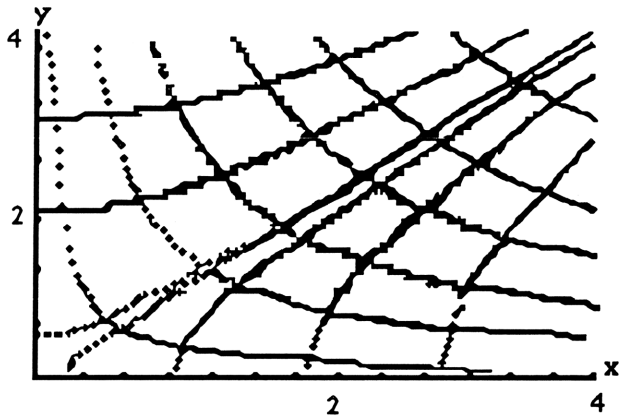
Program to graph trajectories of rabbit and dog	
Stored Quantities	**FN GN H N**
No input on stack	The output is a graph of rabbit and dog paths

```
<< { # 0d # 0d } PVIEW 0 0 0 0 1 R→C 4 ROLLD DRAX
   1 N START IULE2 DUP2 R→C PIXON 4 ROLL
   H 0 R→C + DUP PIXON 4 ROLLD NEXT GRAPH >>
```

(Replace **{ # 0d # 0d } PVIEW** with **CLLCD** and omit the command **GRAPH** in the corresponding HP-28S program.) Use plot parameters to show $-.4 \le x \le 2$, $-.4 \le y \le 1.25$, take **H** = .05 and **N** = 40.

The student should also consider a pursuit problem on the HP-48SX screen in which the rabbit travels around the unit circle and the dog starts from the center. A third pursuit problem is to graph the trajectories of "creatures" who begin at the vertices of a triangle and travel towards their left neighbor with unit speed.

A topic which occurs early in many differential equation textbooks is that of determining trajectories which are orthogonal to the members of a one parameter family of curves, say W(x,y,p) = 0. The usual technique is first to find the differential equation satisfied by the members of the given curve family, say dy/dx = m(x,y); then curves which are orthogonal satisfy dy/dx = -1/m(x,y). If the original family is given in the form dy/dt = f(x,y), dx/dt = g(x,y), trajectories for orthogonal curves satisfy dy/dt = - g(x,y), dx/dt = f(x,y). This latter form is to be



Orthogonal Trajectories dy/dx = -y/x, dy/dx = x/y

preferred if the curves in either family must be specified in terms of a parameter t. Clearly, the programs **IULE2** and **GRXY** can be used to sketch members of both the given family of curves and the orthogonal trajectories. This is our first example of what is called an autonomous system. A specific example is shown above.

Exercise 3.3: Set the plot parameters to show $-.5 \le x \le 3.5$, $-.5 \le y \le 3.5$ and set **N** to 80. Put **FN** to **<< → T X Y 'X*(X^2-Y^2)' >>**, and **GN** to **<< → T X Y 'Y*(3*X^2-Y^2)' >>**. Put **H** = .05 and use **GRXY** to draw a trajectory which starts at (t, x, y) = (0, .5, .1). (This takes slightly over a minute.) Next overlay a trajectory which starts at (t, x, y) = (0, .75, .1). Then reduce **H** to .025 and overlay a trajectory which starts at (t, x, y) = (0, 1, .1). Finally overlay trajectories which start at (t, x, y) = (0, 1.5, .5) and (t, x, y) = (0, 1, .4). At this point we see a family of five oval trajectories. Orthogonal trajectories will result from the **FN** given by **<< → T X Y '-Y*(3*X^2-Y^2)' >>** and **GN** given by **<< → T X Y 'X*(X^2-Y^2)' >>**. Reduce **H** to .01 and overlay trajectories starting at the (t, x, y) points (0, 0, 3.4), (0, 0, 2.5), (0, 0, 1.5), (0, 3, 0), and (0, 2, 0). The result should resemble the figure given in the introduction to this workbook.

Note to HP-28S users: For this exercise and others which require several overlaying graphs constructed using **GRXY**, we create an add graph program. The following program can be used after execution of **GRXY** to enter the first graph. Recall that after **GRXY** the stack contains a string version of the LCD screen on level one. New starting values for t, x, and y are required to start another trajectory so entry into the following program requires string on level 4, start value of t on level 3, start value of x on level 2 and start value of y on level 1. The program is called **AD.G2**.

```
<< CLLCD 4 ROLL DUP 'G1' STO →LCD 1 N START IULE2
DUP2 R→C PIXEL NEXT DGTIZ LCD→ >>
```

Autonomous Problems in Two Dimensions

Graphs in the x-y plane of solutions (x(t), y(t)) of differential equations $x' = f(x,y)$, $y' = g(x,y)$ are called phase plane graphs. If f(x,y) and g(x,y) have continuous partial derivatives, solutions to initial value problems are unique and it is elementary to show that under such circumstances, solution trajectories arising from different initial points either coincide or do not intersect. If fact, it is easy to see that if (x(t), y(t)) is a solution of an equation of this form and a is any constant, then (x(t+a), y(t+a)) is also a solution. Closed trajectories in the phase plane indicate periodic solutions. Constant solutions, that is, points (x,y) such that $f(x,y) = g(x,y) = 0$ are called critical point solutions (also equilibrium solutions). Other trajectories of particular interest are those nearby to a critical point.

- If trajectories arising at all points within some circle around a critical point (x_c, y_c) leave the vicinity of (x_c, y_c) as $t \to \infty$, (x_c, y_c) is called a **repelling solution**, i. e. unstable.

- If trajectories arising at all points within some circle around a critical point (x_c, y_c) approach (x_c, y_c) as $t \to \infty$, (x_c, y_c) is called an **attracting solution**, i. e. asymptotically stable.

Some well studied examples of autonomous are presented below. Note the asymptotic behavior of the solution trajectories as indicated by the graphs.

Example. Systems called **Lotka-Voltera** systems may be scaled to the form

$$dx/dt = x(3 - y), \quad dy/dt = y(x - 3).$$

Such systems arise in the study of populations of two species, one of which feeds on the other. Trajectories which initiate in the first quadrant are periodic. We can obtain graphs by first creating the subprograms

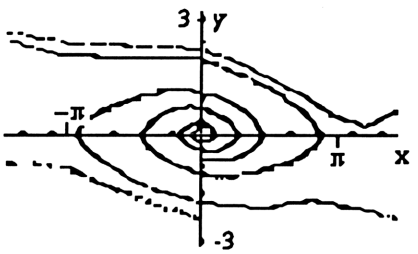
FN: << 3 ROLL DROP 3 SWAP - * >> GN: << 3 ROLL DROP SWAP 3 - * >>

Then we set the plot parameters to show $0 \le x \le 6$, $0 \le y \le 6$, N = 45, and **H** = .05, start at t, x, y = 0, 2, 2 and execute **GRXY**.

Example. The differential equations

$$x'' + cx' + \sin x = 0 \quad \text{or} \quad x' = y,\ y' = -\sin x - cy$$

arise in the study of the displacements of **damped (or undamped) pendulums.** The critical points are (0,0) and ($n\pi$, 0). For $c > 0$, (0, 0) is an attracting solution. Use **GRXY**, c = .3, **FN: << 3 ROLLD DROP DROP>>** and **GN: << 3 ROLL DROP C * SWAP SIN + NEG>>** to obtain the graph shown below. (For c = 0, there is a family of periodic solutions.)



Damped Pendulum Motion (c = .3)

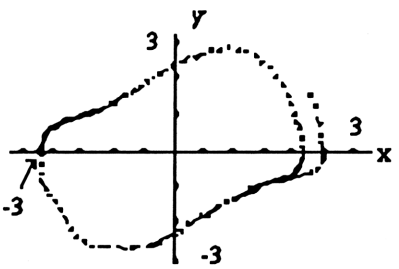
Example. The system

$$dx/dt = -2y + x(1-r^2)/r,\ dy/dt = 2x + y(1-r^2)/r:$$

where ($r^2 = x^2 + y^2$) has an isolated periodic solution r = 1. Here nearby solutions spiral towards the circle r = 1. To obtain graphs use **GRXY** and the functions

```
FN:  << 3 ROLL DROP DUP2 SQ SWAP SQ + DUP √ SWAP 1 -
     NEG SWAP / 3 ROLL * SWAP 2 * - >>
GN:  << 3 ROLL DROP DUP2 SQ SWAP SQ + DUP √ SWAP 1 -
     NEG SWAP / * SWAP 2 * + >>.
```

Another problem which has an isolated attracting periodic solution is the Van der Pol differential equation. This equation was studied in connection with its application to an electronic component. This example in usually studied as a function of a parameter μ contained in the "damping" term. Our figure shows a

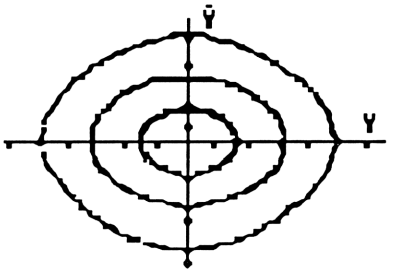


$dx/dt = y,\ dy/dt = -[x + .3(x^2-1)y]$

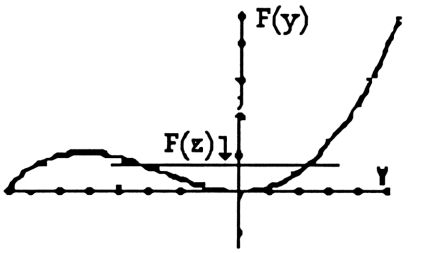
typical graph: here μ = .3. Note the motion is counterclockwise and the solution was started at (x, y) = (2, 2). The solution quickly moves close to its asymptotic shape which is periodic. Solutions starting inside the closed curve (except from (0, 0)) also

move out to the periodic solution. Variation of the parameter μ causes dramatic changes in the shape and period of the solution.

Exercise 3.4: In this exercise we will examine the cycle time of periodic solutions of several special differential equations. The equations under consideration have solutions which resemble the trajectories graphed in the figure below.



Periodic Trajectories



Construction for Trajectories
(See region between F[z] & F[y])

Here we suppose that y(t) satisfies the initial value problem

$$\frac{d^2y}{dt^2} + f(y) = 0, \quad y=(0) = z, \frac{dy}{dt}(0) = 0.$$

where the essential feature of f(y) is that it change sign from negative to positive as y increases thru zero. We multiply by dy/dt and integrate from 0 to t to obtain

$$\frac{dy}{dt} = \pm \sqrt{F(z) - F(y)}\ , \text{ where } F(y) = 2 \int_0^y f(s)\, ds.$$

Suppose we denote by T/2 the time for the trajectory to proceed from the starting point to the state $y(T/2) = z_1$, $dy/dt(T/2) = 0$, then

$$T = 2\int_0^{T/2} dt = 2\int_{z_1}^{z} \frac{dy}{\sqrt{F(z) - F(y)}}.$$

We list the value y_1 for several examples:

(a) $f(y) = y$, $F(y) = y^2$, $z_1 = -z$

(b) $f(y) = y + y^3$, $F(y) = y^2 + y^4/2$, $z_1 = -z$

(c) $f(y) = \sin y$, $F(y) = 2[1 - \cos y]$, $z_1 = -z$

(d) $f(y) = y + y^2$, $F(y) = y^2 + 2y^3/3$, $z_1 =$ largest negative root of $\frac{2}{3}y^2 + \left[1 + \frac{2}{3}z\right]y + \left[z + \frac{2}{3}z^2\right] = 0.$

(e) $f(y) = y + y\cos 4y + .25 \sin 4y$ $F(y) = y^2 + .5y \sin 4y$, $z_1 = -z$

Notice that in (a)-(c) and (e), the function F is even in y, (d) is not. Calculate and plot the values of T for one of the examples (a), (b), (c) or (d) listed above for several values of z. Use the numerical integration key (program) on your calculator with a tolerance 0.005. The following values of T are for part (e) above:

z values	.25	.5	.75	1	1.25	1.5	1.75	2	2.25
T values	3.94	5.29	12.74	21.54	8.29	5.74	5.04	5.45	8.37.

Note $dy/dt = 0$ and $y = \pi/4$ and $dy/dt = 0$, $y = 3\pi/4$ are equilibrium points.

Linear autonomous systems can be solved analytically. These systems have the form:

$$dx/dt = a_{11}x + a_{12}y, \quad dy/dt = a_{21}x + a_{22}y$$

We will consider the case $\det(A) \neq 0$, which means the origin (0,0) is the only critical point. Special solutions have the form $w = \text{column}[x, y] = e^{\lambda t} v$ where λ is

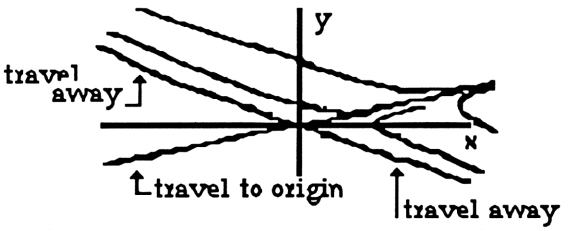
a solution of the equation det (A- λI) = 0 and v will be given below. Such a number λ is called an eigenvalue of the system. The equation det (A- λI) = 0 is called the characteristic equation or the eigenvalue equation for the system. Suppose that λ is an eigenvalue for the system, then the vector v = column [c,d] is a non-zero solution of (A- λI)v = 0. Other solutions of our system are linear combinations of these special solutions (in most cases).

The solution graphs of such systems near the origin (0,0) are particularly interesting. Examples fall into the following cases: closed trajectories (indicating a family of periodic solutions), spiraling trajectories (inward or outward spirals) and curved spoke-like trajectories (again traveling toward or away from the origin). The cases correspond to the type of eigenvalues for the system, viz. purely imaginary values, complex numbers with non-zero real parts and real eigenvalues.

Example: Consider the system

$$dx/dt = x - 4y\,, \qquad dy/dt = -x + 2y.$$

The associated matrix A has eigenvalues $\lambda = .5(3 \pm 17^{.5})$ and corresponding eigenvectors c = column [4, 1.56] and c = column [-4, 2.56]. When a solution starts on a multiple of the first eigenvector, it proceeds toward the origin exponentially. When a solution starts on a multiple of the second eigenvector it travels away from the origin exponentially. Other solutions are a linear combination of these two solutions and eventually proceed away from the origin. Typical trajectories are shown in the figure below. The procedure was to start on the eigenvector solution and trace that trajectory. Other solutions starting very near these special solutions were followed for short periods.



Trajectories near Saddle Point

Exercise 3.3. For the case λ is complex and has negative real part the origin (0,0) is called a **spiral point critical point** (so we have an attracting critical point). Use **IULER2** and **GRXY** to study

$$dx/dt = -.5x + 4y, \quad dy/dt = -4x - .5y$$

with

FN: << 3 ROLL DROP 4 * SWAP .5 * - >>,

GN: << 3 ROLL DROP .5 * SWAP 4 * + NEG >>

Start at (t, x, y) = (0, 0, 1): Set the plot parameters to show $-2 \le x \le 2$, $-1 \le y \le 1$. Use **N** = 120, **H** = .025.

Exercise 3.4. Show the origin (0,0) is neither an attracting or repelling critical point solution for the system x' = -(2x + y), y' = -x + 2y and use **IULE2** to graph the trajectories initiating at (t,x,y) = (0, 0, 1) and at (0, -1, 0). What are the eigenvectors for this systems ? Can you see them on the graphs ?

Solution graphs of <u>nonlinear autonomous systems</u> near a critical point solution can be studied using a linear approximation. Suppose that the vector w = column [x,y] and we have the system dw/dt = F(w) where F(w) = column [f(x,y), g(x,y)] and

where $f(x_c, y_c) = g(x_c, y_c) = 0$. Solution behavior near the critical point $w_c = (x_c, y_c)$ can be determined by studying the linear variational matrix $J(w_c) = F_w(w_c)$ defined below. If all eigenvalues of this matrix have negative real parts, the solution $w = w_c$ is an attracting solution. If one of the eigenvalues has a positive real part, some solutions leave immediate neighborhoods of the critical point. The matrix $J(w_0)$ has i-j element

$$\frac{\partial F_i}{\partial w_j}(w_c)$$

Example: Consider the system $dx/dt = 2x^2 + y^2 - 9$, $dy/dt = x^2 + y^2 - 5$, which has critical point solutions (2,1), (-2,1), (2,-1), (-2,-1). The variational matrix for the last critical point has eigenvalue equation $\lambda^2 + 16\lambda + 8 = 0$. The roots of this equation clearly are negative so that (-2,-1) is an attracting critical point.

Finding critical points is not always easy. Newton's method (introduced in Chapter 2) may be used to find critical points of a system if an approximate location w_o = column $[x_o, y_o]$ of the critical point is known. Then better approximations of the critical point may result from one or more applications of the following algorithm:

$$w_n = w_o - J^{-1}(w_o)F(w_0),\ w_n \rightarrow w_o.$$

We modify the procedure given in Chapter 2 as follows:

1. Put functions $f(x,y)$ $g(x,y)$ in the list **FL**, store **M** = 2, store { X Y } in **PL**, purge **X** and **Y**. Execute **DER** to obtain **JL**= $\{f_x(x,y)\ f_y(x,y)\ g_x(x,y)\ g_y(x,y)\}$.

2. Create matrix **JMAT** with a new program called **JEV** given by

<< JL LIST→ 1 SWAP START →NUM M SQ ROLLD NEXT
{M M} →ARRY 'JMAT' STO >>

3. Create vector **FVEC** with the new program called **FEV** given by

<< FL LIST→ 1 SWAP START →NUM M ROLLD NEXT
{M} →ARRY 'FVEC' STO >>

4. Put an approximation of the critical point **[X Y]** on the stack and execute the new program **NSTP** given by

<< DUP ARRY→ DROP 'Y' STO 'X' STO JEV FEV FVEC JMAT / >>

At this point you have an incremental vector $[X - X_n, Y - Y_n]$ on the stack. If this is sufficiently small, stop with the new vector, which is obtained by the command - (a minus command). If not execute -, then **NSTP** again, etc.

To determine the eigenvalues of the variational matrix we use the programs **DER** and **JEV** given in the previous program to find the matrix $F_w(w_c)$ as follows:

Put functions f(x,y) g(x,y) in the list **FL**, store **M** = 2, store { X Y } in **PL**, purge **X** and **Y**. Execute **DER** to obtain **JL**= $\{f_x(x,y)\ f_y(x,y)\ g_x(x,y)\ g_y(x,y)\}$. Assign values to **X** and **Y**, create **JMAT** with **JEV**.

The eigenvalues satisfy the quadratic equation λ^2 + (**JMAT**[1,1]+**JMAT**[2,2])λ + (**JMAT**[1,1]*__JMAT__[2,2] - **JMAT**[1,2]*__JMAT__[2,1]) = 0. You could use the program **QUAD** as described in the *Hewlett-Packard Reference Manual* for this task.

Exercise 3.5: Find a critical point of the system

$$dx/dt = \sin x + \cos y - x,\ dy/dt = \cos x - \sin y - y$$

near x = 1.9 and y = .2, and determine the eigenvalues of the variational matrix.

(Answer x = 1.9235, y = -.17315, $\lambda = -1.66 \pm i\ .244$)

Adaptive Step Size Method: Autonomous Differential Equations

In many cases, larger step sizes can be taken without loss of accuracy during part of a trajectory, but when sharp changes are encountered, a reduction in step size is needed to maintain accuracy. A straightforward modifications of the adaptive step size Runge Kutta algorithm for a single differential equation can be made. For the problem

$$dx/dt = f(x,y), \quad dy/dt = g(x,y)$$

the algorithm is obtained by considering Taylor series for f(x(t+h), y(t+h)) and g(x(t+h), y(t+h)). Formulas with increments in x, y and t are given by

$$x_{n+1} = x_n + h[2k_1 + 3k_2 + 4k_3]/9, \quad y_{n+1} = y_n + h[2K_1 + 3K_2 + 4K_3]/9$$

$$t_{n+1} = t_n + h$$

where

$$k_1 = f(x_n, y_n), \quad k_2 = f(x_n+hk_1/2, \ y_n+hK_1/2), \quad k_3 = f(x_n +3hk_2/4, \ y_n+3hK_2/4)$$

$$K_1 = g(x_n, y_n), \quad K_2 = g(x_n+hk_1/2, \ y_n+hK_1/2), \quad K_3 = g(x_n +3hk_2/4, y_n+3hK_2/4)$$

and the analysis gives the following estimate for the possible error

$$\text{esterr} \le [\,|\, 2k_1 + 4k_3 - 6k_2 \,|\,+|\, 2K_1 + 4K_3 - 6K_2 \,|\,]/9.$$

Such an algorithm extends easily into the case of n simultaneous differential equations. A program which takes advantage of the vector capability of the calculator has fewer steps and is easy to understand. Thus at this point we consider the differential equation

$$dw/dt = F(w)$$

where w and F are vectors. (In the the two dimension notation used previously, w = [x, y] , F(w) = [f((x,y), g(x,y)].) Again an algorithm for this problem is obtained by considering the Taylor series for F(w(t+h)) which gives the formulas for increments in w and t

$$w_{n+1} = w_n + h[2K(\ ,1) + 3K(\ ,2) + 4K(\ ,3)]/9,\ \ t_{n+1} = t_n + h$$

where

$$K(\ ,1) = F(w_n),\quad K(\ ,2) = F(w_n+hK(\ ,1)/2),\ \ K(\ ,3) = F(w_n\ +3hK(\ ,2)/4)$$

and further analysis gives the following estimate for the possible error

$$\text{esterr} \le [\,|\,|\ 2K(\ ,1) + 4K(\ ,3) - 6K(\ ,2)\ |\,|]/9.$$

The program for the HP-48S/28S given below requires a subprogram **VN** which accepts input vector w and outputs the vector F(w). Following the format used previously for a single differential equation, we store the subprogram **RK3A** which accepts input t w and gives output t w esterr and has stored increments in x in the vector **DELW**. **RK3A** and the single step program **STPN** are given below. The reader should also note a higher order adaptive step size algorithm in Appendix 4 for non-autonomous systems.

Subprogram Name	**RK3A**
Stored Quantities	**VN H**
Input	t and vector w with N components
Output	t w esterr: (delw has been stored)
Command	Resulting stack
<< DUP DUP2	t 4 copies of w
VN	t 3 copies of w K(,1)
DUP 2 *	t 3 copies of w K(,1) 2*K(,1)
4 ROLLD H 2 / *	t w 2*K(,1) w w .5HK(,1)
+	t w 2K(,1) w w+.5HK(,1)
VN DUP 3 *	t w 2K(,1) w K(,2) 3K(,2)
3 ROLLD	t w 2K(,1) 3K(,2) w K(,2)
H .75 * * +	t w 2K(,1) 3K(,2) w +.75HK(,2)
VN 4 *	t w 2K(,1) 3K(,2) 4K(,3)
SWAP	t w 2K(,1) 4K(,3) 3K(,2)
3 DUPN	t w 2K(,1) 4K(,3) 3K(,2) 2K(,1) 4K(,3) 3K(,2)
+ +	t w 2K(,1) 4K(,3) 3K(,2) [2K(,1)+4K(,3)+3K(,2)]
H 9 / *	t w 2K(,1) 4K(,3) 3K(,2) [2K(,1)+ 4K(,3)+3K(,2)]H/9
'DELW' STO	t w 2K(,1) 4K(,3) 3K(,2)
2 NEG *	t w 2K(,1) 4K(,3) -6K(,2)
+ +	t w [2K(,1)+4K(,3)-6K(,2)]
CNRM 9 /	t w estimated error
H 2 / 'H' STO >>	

Subprogram Name	**STPN**
Purpose	take one time step of length, compute new value of x
Input:	t w
Output	t+H new value of w (an adjustment in step size has been made)
Stored Quantities	**RK3A**

```
<< DO RK3A UNTIL .01 < END DELW + SWAP H 2 * +
SWAP H 4 * 'H' STO >>
```

Project exercise in acoustical dynamics. The speed of sound traveling underwater depends on depth. We will use a ray model for underwater acoustic propagation and let z(x) denote the depth of a sound ray at position x, measured along the ocean surface. Snell's law can be written in the form cos θ/C(z) is a constant where tan θ is the slope dz/dx and C(z) denotes the speed of sound transmission at depth z. Change the variable by y = C(z) dz/dx to obtain

$$\frac{dz}{dx} = \frac{y}{C(z)}, \quad \frac{dy}{dx} = -C'(z) .$$

Determine C(z) by a least squares fit of the form $C(z) = a\, e^{-bz} + c + mz$ (a, b, c, and m are constants to be determined) using the data

z	0	500	1000	1500	2000	2500	3000	3500	4000	5000
C(z)	5042	4995	4948	4887	4868	4863	4865	4869	4875	4875
z	6000	7000	8000	9000	10000	11000	12000			
C(z)	4887	4905	4918	4933	4949	4973	4991			

Next, find a value of θ_0 for the initial conditions:

$$z(0) = z_0,\ y(0) = C(z_0) \tan \theta_0 ,$$

so that for $z(x_f) = z_f$, a prescribed number. For this problem take $z_0 = 2000$, $x_f = 24(5280)$, $z_f = 3000$. Plot the ray trajectory.

The function C(z) can be approximated by a least squares fit to data to another type of function (see Forsythe, George, Michael Malcolm and Cleve Moler, *Computer Methods for Mathematical Computations*, Prentice Hall, 1977.) Such a fit is given by

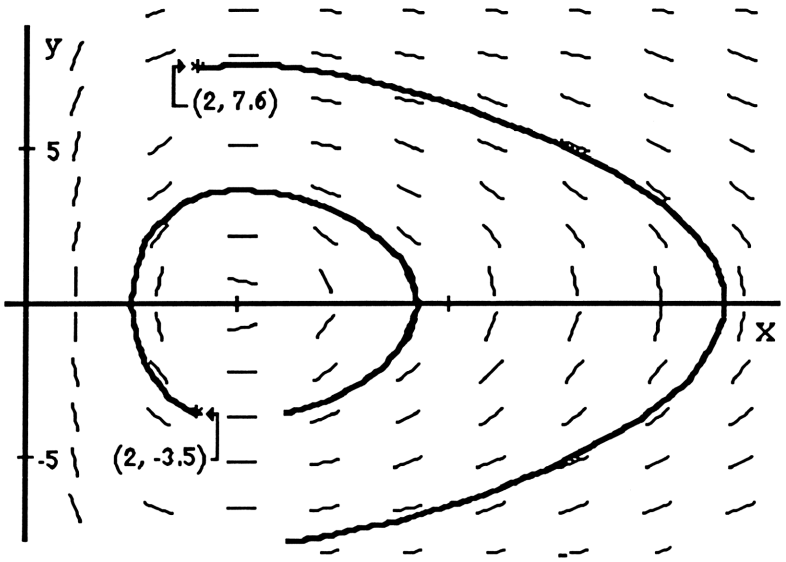
$$C(z) = 4779 + 0.01668\ x + 160{,}295/(x+600).$$

Re-scale the variables x, and z by $t = x/10^4$, $X = z/1000$: the equation become

$$dy/dt = -\ 10\ f'(X),\ \ dX/dt = 100y/f(X)$$

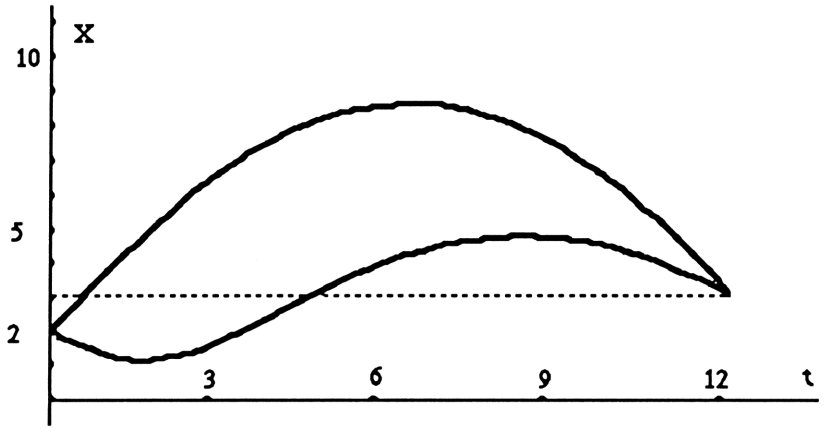
where f(X) = C(1000X). Now we want to find $y(0) = f(2) \tan \theta$ and X(0) = 2 so that X(12.672) = 3. Dividing the differential equations gives $dy/dX = -f'(X)\ f(X)/(10y)$ integration gives $10\ y^2 + f^2(X) =$ a constant. Phase plane graphs are shown. (The adaptive step method was used to obtain the graphs.)

It is interesting to compare the graph of the solution z(x) obtained by using the C(z) given above with that obtained using the C(z) function given in the project exercise above. Any interpolation formula used for C(z) instead of a least squares fit also gives an interesting comparison.



Phase Plot for Acoustic Model

The X vs t graphs are:



Depth versus Horizontal Distance

Discrete Dynamical Systems in the Complex Plane

The discrete dynamical system studied in Chapter 1 had the form $y_{n+1} = F(y_n)$, n = 0, 1, 2, . . . for given y_0. There we studied the behavior of y_n as $n \to \infty$. We consider here the case $z_{n+1} = F(z_n, c)$, n = 0, 1, 2, . . . for given z_0, where all the z elements are complex numbers (a two component vector with special algebraic rules) and the complex parameter number c is given. **The purpose is again to focus attention on the asymptotic behavior of the dynamical system solution sequence** $\{z_n\}$. In particular we will study systems of the form $z_{n+1} = z_n^2 + c$, z_0 given, where c will be fixed in each system. For each number c depending on the starting position z_0 one of three things can happen: (1) $\lim |z_n| = \infty$, (2) $\lim |z_n|$ = some number, or (3) neither of the above. We show the dependance of the elements of the system on the starting value $z_0 = \alpha$ by using the notation $z_n(\alpha)$ for the elements. **The "Julia" set for this sequence** is the boundary of the set $A = \{ \alpha: |z_n(\alpha)| \to \infty \}$. (Elements on the boundary of this set do not belong to the set.) Many of the elements of the Julia set have a wandering property, that is, they do not have a limit and so their behavior is called chaotic. (Iterates $f(\alpha)$, $f(f(\alpha))$, $f(f(f(\alpha)))$, . . . wander around the Julia set.)

The following program is to graph members of the Julia set. At first to establish a procedure of finding lots of members of this set seems tricky because the requirement to be in the set is quite delicate and any roundoff error may cause a sequence α, $f(\alpha)$, $f(f(\alpha))$, $f(f(f(\alpha)))$, . . . beginning with α in the Julia set to drift out of the set. The algorithm to be given is based on the property that for any fixed c, the inverse images of a repelling fixed point belong to the Julia set, that is, for w in the set, the images $f^{-1}(w)$, $f^{-1}(f^{-1}(w))$, $f^{-1}(f^{-1}(f^{-1}(w)))$, . . . also belong to the set. (Here $f^{-1}(w) = \pm\sqrt{(w-c)}$.) It can be shown that the latter sequence is stable for this function f, whereas the sequence of direct images is not stable to roundoff error.

Suppose the mapping $z \rightarrow f(z) = z^2 + c$ has has a fixed point, that is $z^2 + c = z$. There are two such values of z, namely

$$z = \frac{1 \pm \sqrt{1 - 4c}}{2}$$

The student should obtain this number z or several values of the complex number c to see how the HP-28S handles complex square roots and to verify that the absolute value $|f(\text{one of these two values of } z)| > 1$. Such a value of z is called a repelling fixed point of the mapping f. (Complex numbers w near such z have the property that f(w), f(f(w)), f(f(f(w))), . . . get further and further from w.)

We call the repelling fixed point located by z, compute and graph members of the sequence $f^{-1}(z)$, $f^{-1}(f^{-1}(z))$, $f^{-1}(f^{-1}(f^{-1}(z)))$, . . . It can be shown that all such complex numbers satisfy $|w| \leq (1+\sqrt{(1+4|c|)})/2$. For a complex number w since there are two inverse images, viz. $\pm\sqrt{(w-c)}$, the particular sequence of inverse iterates chosen involves a random choice of $\pm$.

Choose and store a complex number **C1**. To set the drawing screen and compute the repelling fixed point z we execute the following subprogram, named **PREP**.

HP-48S version: **<< C1 DUP ABS 4 * 1 + √ 1 + 2 / DUP NEG SWAP DUP2 XRNG YRNG 4 * 1 SWAP - √ 1 + 2 / >>**

HP-28S version: **<< C1 DUP ABS 4 * 1 + √ 1 + 2 / DUP 2 * SWAP R→C DUP NEG PMIN R→C PMAX 4 * 1 SWAP - √ 1 + 2 / >>**

The output is a fixed point of f. If the output has absolute value larger than .5 call the number Z, if not put Z = 1 - the output. Store **Z** and execute the program **BACK** given by

HP-48S version: **<< { # 0d # 0d } PVIEW Z BCK1 >>,**

HP-28S version **<< CLLCD Z BCK1 >>**

Here **BCK1** for the HP-48S is

<< 1 500 START C1 - √ ONE * DUP PIXON NEXT DROP GRAPH >>

The HP-28S version is

<< 1 500 START C1 - √ ONE * DUP PIXEL NEXT DROP LCD→ >>

and the subprogram **ONE** is given by

<< (1,0) IF RAND .5 < THEN NEG END >>.

For c = (-.12256, .7449), the following "graph" results

Douady's Rabbit: c = (-.123, .745)

Exercise 3.6: Execute the program sequence given above for c = (-1,0), c = (-.5, -.1).

Chapter 4. Higher Order Systems

In this chapter we consider linear systems of differential equations of the form $y' = Ay + f(t)$ where y and f(t) are vectors with, say n components, and A is an n by n matrix. Following this we give some introduction to nonlinear systems. In particular we give some examples where the "eigenvalues" of the linear systems which result when a nonlinear problem is linearized around a critical point can be used to specify whether the critical point solution is repelling, attracting, or neither. For linear homogeneous systems of differential equations we can use the graphing feature of the calculator to solve for real roots of the nth order polynomial "eigenvalue" (or characteristic) equation. The real roots may be factored out of the polynomial and a program given by Hewlett Packard can be used to find other roots. The linear system solver is quite useful in determining constants in the solution of linear differential equations when initial values are given. We will give a program which allows us to find solutions of the singular systems which result when "eigenvalues" and corresponding "eigenvectors" are determined. We will use the calculator's ability to find derivatives to determine the characteristics of critical point solutions of nonlinear systems. To find critical points may require the solution of several algebraic equations: a job for Newton's method again. We will only introduce the matrix algebra in these notes. A more complete discussion is contained in *Calculator Enhancement for a Linear Algebra Course* by D. R. LaTorre and published by Harcourt Brace and Jovanovich.

Solutions for $y' = Ay + f(t)$

Consider first the vector problem $dy/dt = Ay$. Here we want to find all solutions of the differential equation. It is readily shown that if n independent vector

functions which satisfy the differential equation can be determined and placed in the columns of a matrix Y(t), then all solutions have the form Y(t)c where c is a vector with n components. The "educated guess" $y(t) = e^{\lambda t}c$ (here y(t) and c are vectors) leads to the nth order polynomial equation $\det(A-\lambda I) = 0$ which is called the eigenvalue or characteristic equation, and to the problem of determining nontrivial solution vectors c to the problem $(A-\lambda I)c = 0$ (where λ is a solution to the eigenvalue equation). Thus the problem breaks into several parts: (1) find the eigenvalue equation, (2) find the solutions of the eigenvalue equation, (3) for each solution λ, find a corresponding eigenvector c, and (4) assemble the matrix Y(t). We will illustrate the solution process first for n = 3 component systems, then give two programs which will assist in the higher dimensional situations. Examples will be given.

A calculator program **to display the eigenvalue equation** in the 3 by 3 case is:

Program Name	**EIG3** HP-48S version
Purpose:	Display the eigenvalue equation
Stored Quantities:	3 by 3 matrix A

```
<< 'X' PURGE A DET 8 RND → DT1 << 'X^3 - (A(1,1) +
  A(2,2) + A(3,3))*X^2 + (A(1,1)*A(2,2) - A(1,2)*A(2,1) +
A(1,1)*A(3,3) - A(1,3)*A(3,1) + A(2,2)*A(3,3) - A(3,2)*A(2,3))*X -
                    DT1' EVAL >> >>
```

For the HP-28S replace **8 RND** with **8 FIX RND STD** The program as given will display the eigenvalue equation as a cubic in x. (Note: make sure x is not defined in the **HOME** directory.)

Exercise 4.1: Find the characteristic equation for the matrices

$$A=\begin{bmatrix} 1 & 2 & -1 \\ 1 & 0 & 1 \\ 4 & -4 & 5 \end{bmatrix}, \quad A=\begin{bmatrix} -1 & -6 & 3 \\ 3 & 8 & -3 \\ 6 & 12 & -4 \end{bmatrix}, \quad A=\begin{bmatrix} -5 & -8 & -12 \\ -6 & -10 & -10 \\ 6 & 10 & 13 \end{bmatrix}.$$

(The first matrix has the eigenvalue equation x^3 - 6x^2 + 11x - 6.)

To find the roots of the eigenvalue equation isolate one root or more roots by storing the equation (**STEQ**) and using the **DRAW** and **SOLVR** programs. You may have to try several settings of the plot parameters (**XRNG, YRNG** on the HP-48S or **PMIN** and **PMAX** on the HP-28S).

Exercise 4.2: Find the eigenvalues of the matrices given in exercise 4.1. (Eigenvalues for the first matrix are 1, 2, 3.)

Consider the matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

The eigenvalue equation is x^3 - x - 1. A zero of this equation obtained by the **SOLVE** routine is x = 1.3247---. If we divide x - 1.3247--- into x^3 - x - 1 we obtain the quotient x^2 + 1.3247---x +(1.3247--^2-1). Zeros of this quadratic are complex eigenvalues. At this point the x has a value stored in it. To avoid confused notation we take an extra step: bring the value in x to the stack and store it in R. Now place 'x^2 + r*x +(r^2-1)' on the stack, **'X' PURGE** and **EVAL**. You now have the desired quadratic on the stack, enter **'X'** and execute **QUAD** (on the **ALGEBRA** menu for the HP-48S: on the **SOLVE** menu on the HP-28S). Follow the usual procedure for the **QUAD** program to obtain the roots -.662 ± i .563.

Exercise 4.3: Determine the eigenvalues for

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 4 & 3 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 3 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} -11 & -8 & -12 \\ 2 & 1 & 4 \\ 6 & 4 & 5 \end{bmatrix}$$

When an eigenvalue λ is determined, the matrix $(A-\lambda I)$ is singular and the linear system solver is not appropriate to solve the equation $(A-\lambda I)c = 0$. Place $(A-\lambda I)$ on the stack and use the programs named **PIV** and **ROKL** to obtain the Gauss-Jordon echelon form to determine the row space of $(A-\lambda I)$ and nontrivial solution vectors c.

Program Name	**PIV**	HP-48S version
Purpose	Gauss pivot on element **K L**	
Input: Matrix **A**, integers **K L**		Output: Altered matrix **A**

```
<< → A K L << IF 'A(K,L)' EVAL 0 == THEN "PIVOT ENTRY
   IS 0" ELSE A SIZE 1 GET → M << M IDN 'A(1,1)'
      EVAL TYPE IF THEN DUP 0 CON R→C END
   1 M FOR I 'A(I,L)' EVAL { I K } SWAP PUT NEXT INV
         A * >> 8 RND END >> >>
```

For the HP-28S calculator replace **8 RND** with **8 FIX RND STD**

Program Name	**ROKL**	
Purpose	Interchange rows **K** and **L**	
Input: Matrix **A**, integers **K L**		Output: Altered matrix **A**

```
<< → A K L << A SIZE 2 GET → N << A 1 N FOR I
'A(K,I)' EVAL { L I } SWAP PUT NEXT 1 N FOR J 'A(L,J)'
      EVAL { K J } SWAP PUT NEXT >> >> >>
```

Notice that the programs **PIV** and **ROKL** given above are valid for any size square matrix.

Example: The first matrix in the exercise 4.1 has eigenvalues 1, 2, and 3.

For $\lambda = 1$ the equation for c is

$$\begin{bmatrix} 0 & 2 & -1 \\ 1 & -1 & 1 \\ 4 & -4 & 4 \end{bmatrix} c = 0.$$

If this matrix is placed on the stack and the command 1, 2 **ROKL** is given we get

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 2 & -1 \\ 4 & -4 & 4 \end{bmatrix};$$

now give the command 1,1 **PIV** to get

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix};$$

now 2,2 **PIV** gives

$$\begin{bmatrix} 1 & 0 & .5 \\ 0 & 1 & -.5 \\ 0 & 0 & 0 \end{bmatrix}.$$

The solution relations $c_1 = -.5\, c_3$, $c_2 = .5\, c_3$ result: i. e., c = [-1, 1, 2] or any nonzero multiple of this vector. For $\lambda = 2$, we obtain that any multiple of c = [-2, 1, 4] is a corresponding eigenvector; for $\lambda = 3$, we obtain that any multiple of c = [-1, 1, 4] is a corresponding eigenvector.

Example: The matrix

$$A = \begin{bmatrix} 6 & 7 & 8 \\ -2 & 0 & -2 \\ -4 & -6 & -6 \end{bmatrix}$$

has eigenvalues $\lambda = -2$ and $1 \pm i$. The procedure shown above gives the eigenvector c= column [1, 0, -1] corresponding to $\lambda = -2$. For $\lambda = 1 + i$, the matrix $A - \lambda I$ is

$$A = \begin{bmatrix} (5,-1) & 7 & 8 \\ -2 & (-1,-1) & -2 \\ -4 & -6 & (-7,-1) \end{bmatrix}.$$

When we use 1 1 **PIV**, then 2 2 **PIV** we obtain

$$\begin{bmatrix} (1,0) & (0,0) & (1,-.5) \\ (0,0) & (1,0) & (.5,.5) \\ 0 & 0 & 0 \end{bmatrix}.$$

This leads to the eigenvector c = column [(-1,.5), ((-.5,-.5), 1]. Recall for the conjugate eigenvalue, there is a eigenvector conjugate to this c vector.

The next step is to assemble a fundamental matrix of solutions Y(t) which has columns the vector solutions determined above. For the first matrix in exercise 4.1 we have determined eigenvalues and corresponding eigenvectors in the example just after the **ROKL** program. We have

$$Y(t) = \begin{bmatrix} -e^t & -2e^{2t} & -e^{3t} \\ e^t & e^{2t} & e^{3t} \\ 2e^t & 4e^{2t} & 4e^{3t} \end{bmatrix}.$$

Exercise 4.4: Now give the solution of y' = Ay: with conditions: $y(0) = [1 \;\; 3 \;\; -5]^t$.

For the matrix example given just above the preceding paragraph (one real and a pair of complex eigenvalue) we proceed as follows. If a matrix A has eigenvalues

$\lambda = \alpha \pm \beta i$ and corresponding eigenvectors $c = a \pm ib$, by adding the exponential solutions obtained it is known that the quantities

$$e^{\alpha t} (\cos \beta t\, a - \sin \beta t\, b) \text{ and } e^{\alpha t} (\sin \beta t\, a + \cos \beta t\, b)$$

are real valued solutions of the differential equation $y' = Ay$. Consequently for this example we get the fundamental matrix of solutions

$$Y(t) = \begin{bmatrix} -e^t (\cos t + .5 \sin t) & e^t (-\sin t + .5 \cos t) & e^{-2t} \\ .5e^t (-\cos t + \sin t) & -.5e^t (\sin t + \cos t) & 0 \\ e^t \cos t & e^t \sin t & -e^{-2t} \end{bmatrix}.$$

Exercise 4.5: Find a fundamental matrix of solutions of $dy/dt = Ay$ for

$$A = \begin{bmatrix} -4 & -4 & -5 \\ -1 & -1 & -1 \\ 4 & 4 & 5 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 4 & 3 & 0 \end{bmatrix}.$$

When there is a eigenvalue λ of multiplicity two, either there are two independent eigenvectors c such that $(A - \lambda I)c = 0$ or there is a solution of the form $y(t) = e^{\lambda t} (ct + d)$. c will be an eigenvector and $(A - \lambda I)^2 d = 0$. For

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

$\lambda = -2$ is a eigenvalue of multiplicity 2 and $\lambda = -1$ is a simple eigenvalue. The eigenvectors corresponding to $\lambda = -2$ are multiples of c = column [1, -1, 2] and the eigenvectors corresponding to $\lambda = -1$ are multiples of c = column [1, -1, 3]. The equation $(A+2I)^2 d = 0$ has a solution d = column [0, 1, 0]. (Such a vector is easily obtained on the calculator, first by calculating $(A+2I)^2$, then using **PIV** to obtain d.) For this matrix A we have a fundamental matrix of solutions

$$Y(t) = \begin{bmatrix} e^{-t} & e^{-2t} & te^{-2t} \\ -e^{-t} & -e^{-2t} & (1-t)e^{-2t} \\ 3e^{-t} & 2e^{-2t} & 2te^{-2t} \end{bmatrix}.$$

For the matrix

$$A = \begin{bmatrix} -5 & -2 & -3 \\ 0 & -3 & 0 \\ 2 & 2 & 0 \end{bmatrix}$$

$\lambda = -3$ is a eigenvalue of multiplicity 2 and $\lambda = -2$ is a simple eigenvalue. The eigenvectors corresponding to $\lambda = -3$ are linear combinations of c = column [1, -1, 0] and c = column [-3, 0, 2]. The eigenvectors corresponding to $\lambda = -2$ are multiples of c = column [1, 0, -1]. A fundamental matrix of solutions is

$$Y(t) = \begin{bmatrix} e^{-2t} & e^{-3t} & -3e^{-3t} \\ 0 & -e^{-3t} & 0 \\ -e^{-2t} & 0 & 2e^{-3t} \end{bmatrix}.$$

Exercise 4.6: Find a fundamental matrix of solutions for the system y' = Ay where

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} -3 & 1 & 0 \\ 0 & -3 & 1 \\ 4 & -8 & 2 \end{bmatrix}.$$

We wish to present a program which accepts an n by n matrix as input and generates its eigenvalue equation. There is a algorithm for the coefficients of this equation which combines many subdeterminants to form the coefficients. Such an algorithm seems cumbersome for the HP-48/28S; however another less well known algorithm involves products and sums of n by n matrices and the computation of the trace of some of these matrices, something this calculator does with little trouble. The following algorithm is taken from Cullen, *Linear Algebra with Applications*, Scott

Foresman and Company, 1988: Let A be an n by n matrix, set $B_0 = I$ and then for k = 1, 2, . . ., n let

$$A_k = AB_{k-1},\ c_k = -(1/k)\ tr(A_k),\ B_k = A_k + c_kI.$$

Then the characteristic polynomial is given by

$$\lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \ldots + c_{n-1}\lambda + c_n.$$

Program Name	**CHAR**	HP-48S version
Purpose	Find the eigenvalue equation for a matrix in level 1 on the stack	
Input stack: square matrix		Output stack: list of the coefficients in characteristic equation

```
<< DUP SIZE 1 GET { 1 } → mtx n poly << mtx 1 n FOR j
    0 1 n FOR k OVER { k k } GET + NEXT
  j NEG / 'poly' OVER STO+ mtx DUP ROT * SWAP ROT *
    + NEXT DROP poly >> >>
```

For HP-28S, remove the ' marks from **'poly'** and replace **STO+** with + **'poly' STO**

Example: Place the matrix

$$A = \begin{bmatrix} 3 & 4 & 6 & 4 \\ 1 & -1 & 1 & 1 \\ 6 & 8 & 7 & 8 \\ -11 & -12 & -15 & -14 \end{bmatrix}$$

on the stack and execute **CHAR**. You should receive output { 1 5 13 19 10 } meaning the eigenvalue equation is $\lambda^4 + 5\lambda^3 + 13\lambda^2 + 19\lambda + 10 = 0$. This equation has two real zeros, $\lambda = -1$ and $\lambda = -2$. Dividing $\lambda^2 + 3\lambda + 2$ into the eigenvalue equation gives a factor $\lambda^2 + 2\lambda + 5$ so $\lambda = -1 \pm i$ are two remaining eigenvalues. The procedure given

above for 3 by 3 matrices extends to n by n matrices and therefore a fundamental matrix of solutions is

$$Y(t) = \begin{bmatrix} 4e^{-t} & 0 & e^{-t}\cos 2t & e^{-t}\sin 2t \\ e^{-t} & e^{-2t} & 0 & 0 \\ -2e^{-t} & 0 & -e^{-t}\sin 2t & e^{-t}\cos 2t \\ -2e^{-t} & -e^{-2t} & e^{-t}(\sin s\, t - \cos 2t) & -e^{-t}(\cos 2t + \sin 2t) \end{bmatrix}.$$

Exercises 4.7: Find a fundamental matrix of solutions for $y' = Ay$ when

$$A = \begin{bmatrix} -2.5 & 2.5 & -3.5 & -.5 \\ 1 & 2 & 2 & 1 \\ 5 & 4 & 6 & 6 \\ -4.5 & 8.5 & -5.5 & -7.5 \end{bmatrix}$$

The booklet, *Mathematical Applications, Step by Step Solutions for your HP-28S Calculator*, published by Hewlett-Packard contains a sequence of programs to solve polynomial equations. By combining such programs with the material given above, the student should be able to determine many solutions to a system $dy/dt = Ay$.

To obtain solutions of the nonhomogeneous equation $y' = Ay + f(t)$, suppose that a fundamental matrix $Y(t)$ of solutions for the associated homogeneous equation is known (so $Y'(t) = AY(t)$). It is easy to see that

$$y(t) = Y(t)\, c + \int_0^t Y(t - s)\, Y^{-1}(0)\, f(s)\, ds$$

is a solution for any vector c. If initial conditions are known we may determine c. In the general case a program which uses the numerical integration capability of the calculator can produce values at various times t for the components of the integral listed above. If the functions in f(t) are elementary we can use the method of undetermined coefficients to construct a particular solution. Of course this technique

will require the solution of linear algebraic equations to compute the coefficients, an easy step on this calculator.

Euler Algorithm for Vector Equations

The following program is for one step in the solution of

$$dw/dt = F(t, w),$$

with suitable initial conditions. Here w and F(t,w) are vectors. We assume that that the step size H is stored and there is a subprogram **VN** which takes t w from the stack and returns F(t, w). The program begins with t w on the stack and terminates with the new values of t w after one step on the stack.

Program Name	**EULN**		
Purpose	generate new values of t w (vector) resulting from one step using the Euler algorithm		
Stored Quantities	**H VN**		
Input		Output	
level 2	level 1	level 2	level 1
t	w	t_{new}	w_{new}

<< DUP2 VN H * + SWAP H + SWAP>>

Notice the program is an exact copy of **EULER** (chapter 1) except **VN** replaces **FN**. The program **IULER** should be changed to treat vector problems and called **IULN**.

Pursuit Problem: A rabbit starts at (0,1) and runs along y =1 with speed 1. At the same time a dog starts at (0, 0) and pursues the rabbit with speed 1.3. The dog attempts to point at the rabbit at all times but is constrained by his momentum.

That is, for the angle z between the dog's direction and the x axis, dz/dt is restricted. What is the path of the dog ? The equations of motion are:

$$dx/dt = 1.3 \cos z, \quad dy/dt = 1.3 \sin z, \quad dz/dt = -(z - \theta(t,x,y))$$
$$x(0) = 0, \quad y(0) = 0, \; z(0) = 2.2.$$

Here $\theta(t,x,y)$ is the angle between the dog-rabbit vector and the x axis. We take **H** = .05, **N** = 120, plot parameters to show $-1 \le x \le 6$, $-.5 \le y \le 2$, repeatedly apply **EULN** and watch the trajectory. Suitable functions for f(t, x, y, z), g(t, x, y, z), and H(t,x,y) (here F(w) = column [f, g, H]) are given by:

```
FN:   << 4 ROLLD DROP DROP DROP COS 1.3 * >>
GN:   << 4 ROLLD DROP DROP DROP SIN 1.3 * >>

HN:   << 4 ROLLD 1 - DUP 4 ROLLD SQ 3 ROLLD - DUP 3
      ROLLD SQ + √ 3 ROLLD DUP 3 ROLLD THTA - NEG>>
      THTA: << 0 IF ≥ THEN THT1 ELSE THT2 END >>

      THT1: << 0 IF ≤ THEN SWAP / ACOS ELSE
            SWAP / ACOS NEG END >>

THT2: << 0 IF < THEN NEG SWAP / ACOS π →NUM
      SWAP - ELSE NEG SWAP / ACOS π →NUM + END >>

VN: << DUP2 DUP2 ARRY→ DROP FN 5 ROLLD ARRY→ DROP GN
      4 ROLLD ARRY→ DROP HN 3 →ARRY >>
```

An easy modification of the **GRAF** program can be used to follow the action. The program requires a starting stack of 0, [0, 0, 1.8] and the trajectories of both dog and rabbit are shown dynamically as follows:

HP-48S version:

```
<< { # 0d # 0d } PVIEW 0 1 R→C 3 ROLLD DRAX 1 N START IULN
DUP OBJ→ DROP2 R→C PIXON 3 ROLL H 0 R→C + DUP PIXON 3
ROLLD NEXT GRAPH >>
```

For the HP-28S version replace **{ # 0d # 0d } PVIEW** with **CLLCD**, replace **OBJ→** with **ARRY→** , replace **PIXON** with **PIXEL** (twice) and replace **GRAPH** with **DGTIZ LCD→**

A Nonlinear Problem: the Lorentz Equations

Consider the problem:

$$dx/dt = \sigma(y-x),\ dy/dt = (r-z)x - y,\ dz/dt = xy - bz.$$

This set of equations was studied by E. Lorentz (1963) in connection with convective heat transfer between the earth's surface and the atmosphere. A point (x,y,z) represents convection velocities and temperature profile, vertical and horizontal. An equilibrium or periodic solution to the system represents predictable behavior. The graphs of particular solutions gave particularly surprising results because most trajectories never seem to approach such predictability. The paper has been one of the seminal studies in the area of chaos.

First the three critical points (that is, points (x, y, z) where the right sides of the differential equations are zero) are (0, 0, 0) and $(\pm(b(r-1))^{.5}, \pm(b(r-1))^{.5}, r-1)$.

The variational matrix at (0, 0, 0) is $\begin{bmatrix} -\sigma & \sigma & 0 \\ r & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$, with system eigenvalues are -b and $.5(-(\sigma+1) \pm \sqrt{(\sigma-1)^2 + 4\sigma r}\ )$.

The variational matrices near the remaining critical points are

$$\begin{bmatrix} -\sigma & \sigma & 0 \\ 1 & -1 & -\delta \\ \delta & \delta & -b \end{bmatrix} : \delta = \pm\sqrt{b(r-1)} \ .$$

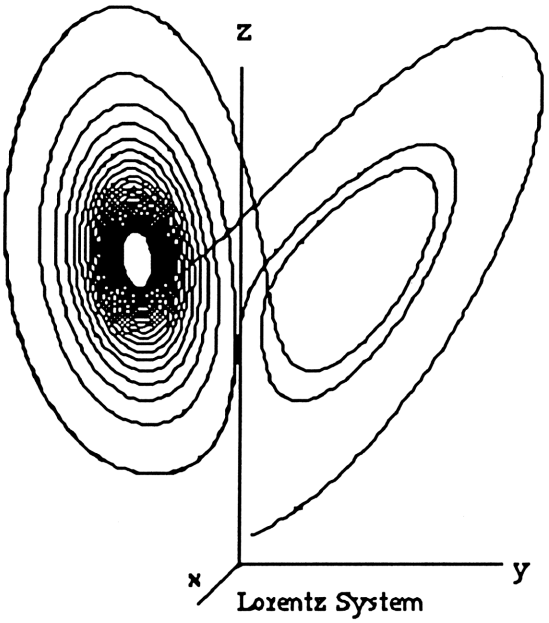
Here eigenvalues satisfy the equation

$$\lambda^3 + (\sigma+b+1)\lambda^2 + b(r+\sigma)\lambda + 2\sigma b(r-1) = 0 \ .$$

For $r > 1$, $b>0$, $\sigma > 0$ one of the variational eigenvalues near (0, 0, 0) is positive, so most trajectories starting near (0, 0, 0) leave that neighborhood. For r sufficiently near one, the eigenvalues of the remaining variational matrices are negative and the corresponding critical points are attracting solutions. When r is slightly larger, one of the variation eigenvalues has a positive real part and the critical point is repelling. It can be shown that solutions starting near the critical points stay bounded and thus the trajectories have interesting behavior as t increases.

Exercise 4.8: Compute a solution for initial values $x(0) = 13.25$, $y(0) = 19$, $z(0) = 26$ for $\sigma = 10$, $r = 28$, and $b = 8/3$ using the program **STPN** given in chapter 3 and using initial value of $H = .01$ for $0 \le t \le 1$. Part of a typical trajectory is shown below. A continuation of the trajectory reveals no asymptotic pattern other than a continual looping in two of the x, y, z octants. Your initial point occurs after the trajectory has looped several times in a negative x, y octant and briefly returned to the positive x, y octant. The following table shows partial results rounded to three decimals. (The first entry shows how many applications to **STPN** were applied to get the result.)

Iter #	time	x	y	z
50	.0313	14.709	17.996	31.799
100	.0625	15.145	14.482	36.646
150	.0938	14.271	9.441	39.490
200	.125	12.269	4.601	38.711
250	.1563	9.688	1.158	36.533
300	.2181	4.914	-1.477	30.805
350	.3431	0.412	-1.161	21.745



Lorentz System

A Nonlinear Problem: Earth, Moon, Satellite Motion

Consider the model problem where the moon is circling the earth and a satellite is in motion in the plane of the earth-moon orbit. The "restricted three body" problem results when the satellite mass can be ignored when compared to the masses of the moon and earth. (See *Celestial Mechanics*, Part II by S. Sternberg, W. A. Benjamin Company, 1969) If a rotating coordinate system is used so the coordinates of the earth are $(-\mu,0)$ and the coordinates of the moon are $(1-\mu,0)$ and the coordinates of the satellite are denoted by $(x(t), y(t))$, the equations of motion are

$$\frac{dx}{dt} = u, \quad \frac{du}{dt} = 2v + x - \frac{(1-\mu)}{r^3}(x+\mu) - \mu\frac{(x+\mu-1)}{\rho^3}$$

$$\frac{dy}{dt} = v, \quad \frac{dv}{dt} = -2u + y - \frac{(1-\mu)}{r^3}y - \mu\frac{y}{\rho^3}$$

where

$$r^2 = (x+\mu)^2 + y^2, \quad \rho^2 = (x+\mu-1)^2 + y^2$$

The constant μ is a ratio of the masses of the earth and moon ($= m_p/(m_p+m_m) = 1/82.45$). For the "earth-moon" system it has been discovered that a solution with period approximately 6.1922 results from the initial conditions

$$x(0) = 1.2 \qquad u(0) = 0, \qquad y(0) = 0, \qquad v(0) = -1.04936...$$

We take the vector w = [x, u, y, v] and create the subprogram

Subprogram Name	**RN**	HP-48S version
Stored Quantities	none	
input w:		output: **R1** = r^3 **R2** = ρ^3

```
<< OBJ→ DROP2 SWAP DROP SQ SWAP 82.45 INV +
    DUP2 SQ + 1.5 ^ 3 ROLLD 1 - SQ + 1.5 ^ >>
```

For the HP-28S version, replace **OBJ→** with **ARRY→** Then **VN** which takes w to F(w) for the HP-48S can be as follows:

```
<< DUP RN 82.45 INV → R1 R2 MU << OBJ→ DROP 3 ROLLD MU DUP
 R2 / SWAP 1 - R1 / - 1 - NEG * SWAP DUP 5 ROLLD 2 * - 3 ROLLD
SWAP DUP MU + 1 - R2 / MU * SWAP DUP 3 ROLLD MU + 1 MU - R1
   / * + - SWAP DUP 2 * 3 ROLL + 3 ROLLD SWAP 3 →ARRY >> >>
```

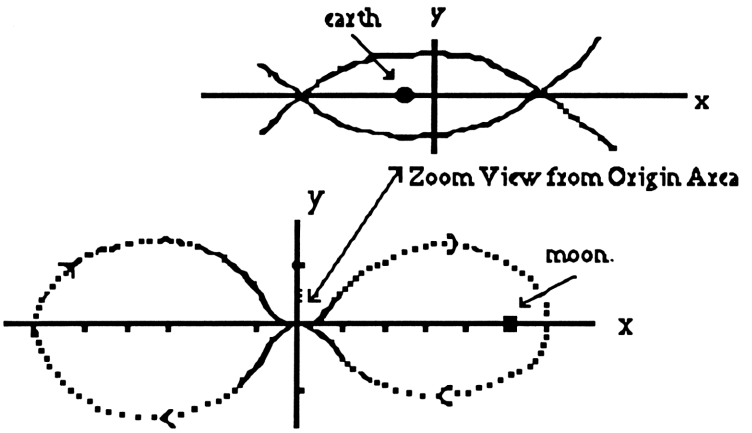
Replace **OBJ→** with **ARRY→** for the HP-28S version.

Since this program is complex, we provide a stack status at various program steps:

Instruction	Stack contents
DUP RN 82.45 INV	w R1 R2 MU
→ R1 R2 MU	w
OBJ→ DROP	x u y v
3 ROLLD MU DUP R2 /	x v u y μ μ /R2
SWAP 1 - R1 /	x v u y μ /R2 (μ −1)/R1
- 1 - NEG *	x v u y*(-1)[μ /R2-(μ −1)/R1 -1]
SWAP DUP 5 ROLLD 2 * -	u x v {y*(-1)[μ /R2-(μ −1)/R1 -1]-2u}
3 ROLLD SWAP	u last equation v x
DUP MU + 1 - R2 / MU *	u last equation v x μ[x+μ−1]/R2
SWAP DUP 3 ROLLD	u last equation v x μ[x+μ−1]/R2 x
MU + 1 MU - R1 /	u last equation v x μ[x+μ−1]/R2 x+μ (1-μ)/R1
*** + - SWAP DUP**	u last equation x-μ[x+μ−1]/R2+(x+μ) (1-μ)/R1 v v
2 * 3 ROLL + 3 ROLLD SWAP →ARRY >>	

The figure shown below results from an initial tolerance setting in the **STPN** program of 0.01 and the initial conditions given above with adjustments in the tolerance as follows. As the satellite approaches the earth, the step size becomes

small, so the tolerance is increased while the satellite is near the earth: later it is reduced back to 0.01, etc.



Satellite near Earth-Moon System

Another interesting periodic solution occurs for the approximate initial conditions

$$x(0) = 0, \qquad u(0) = 1.58, \qquad y(0) = 1.2, \qquad v(0) = 0.$$

We note equilibrium points for the restricted three body satisfy

$$u = v = 0, \; y\,[\,1 - (1-\mu)/r^3 - \mu/\rho^3] = 0, \; x = (1-\mu)(x+\mu)/r^3 + \mu(x-1+\mu)/\rho^3.$$

Project: Find the equilibrium solutions of this system and determine their stability properties. Notice that for $y = 0$, we need only solve an equation of the form $g(x) = 0$, but for other solutions, we need to find the solution of a pair of nonlinear algebraic equations. We can use Newton's the procedure presented in Chapter 3 for finding such values of x and y if we can find a suitable starting point x_0, y_0. Appendix 3 outlines a method to do this using the calculator. (Two of the equilibrium points are located at $x = .48787$, $y = \pm .86603$.) To determine the

stability, we need to find the eigenvalues of the 4 by 4 variational matrices associated with each critical point solution.

Parameter Identification Problems Revisited

In chapter 1, we studied several population models containing space/food limitations on growth. See exercises 1.4, 1.7, 1.8 and 1.9. In chapter 2, we showed how to choose problem parameters to achieve a fit to data observations when the functional form of the solution g(t, p, q) is known. New problem: choose p and q so that the solution of

$$dy/dt = q\, y\, (1 - y^p),\ \ y(0) = .2$$

best fits the (t, y) data (1, .4), (2, .5), (3, .75), and (4, .9).

Here we have an initial value problem, say for population growth, of the form $dy/dt = f(y,p,q)$, $y(0) = y_0$ and wish to choose the parameters p and q so that a close fit to data is achieved. The solution must be obtained by using a numerical method (improved Euler, Runge Kutta, etc.) and the functional form of the solution is unknown. Even though we may attack a vector initial value problem of this type, for simplicity we will assume y, p, and q are real numbers and for values of y, p, q in the domain of f, f(y, p, q) is a real number. If we also assume f is a smooth function, we can differentiate the differential equation with respect to p to obtain

$$\frac{du}{dt} = f_y\,(y, p, q)\, u + f_p\,(y, p, q)$$

where $u = \partial y/\partial p$ and f_y and f_p denote the partial derivatives of f with respect to y and p respectively. A similar equation holds for $v = \partial y/\partial q$. Consider the vector initial value problem $dw/dt = F(w)$ with $w = w(0)$ at $t = 0$ for

$$w = \begin{bmatrix} y \\ u \\ v \end{bmatrix}, \quad F(w) = \begin{bmatrix} f(y, p, q) \\ f_y(y, p, q)\, u + f_p(y, p, q) \\ f_y(y, p, q)\, v + f_q(y, p, q) \end{bmatrix}, \quad w(0) = \begin{bmatrix} y_0 \\ 0 \\ 0 \end{bmatrix}.$$

Our criterion for best fit is to choose p, q to minimize the payoff function

$$J = \sum_{i=1}^{N} [y_i - y(t_i, p, q)]^2.$$

Here the data observations are $\{ (t_1, y_1), (t_2, y_2), \ldots, (t_N, y_N)$,and $y(t, p, q)$ is the solution of the original problem (and the first component of the solution of the vector equation). If we set the derivatives of J with respect to p and q to zero, we get

$$\sum_{i=1}^{N} [y_i - y(t_i, p, q)]\, u(t_i, p, q) = 0, \quad \sum_{i=1}^{N} [y_i - y(t_i, p, q)]\, v(t_i, p, q) = 0.$$

to determine p and q.

Our first thought here may be to apply Newton's method to determine solutions p, q of these two nonlinear equations. However, recall that Newton's method would require partial derivatives of u and v with respect to p and q. This could be done by differentiating the original differential equation more times, but then we would need to solve an initial value problem containing 6 differential equations !

Alternately, suppose that we have a trial set of parameters p and q and wish to choose better values $p + \Delta p$, $q + \Delta q$. If the incremental values are small then

$$y(t_i, p+\Delta p, q+\Delta q) \approx y(t_i, p, q) + \frac{\partial y}{\partial p}(t_i, p, q)\, \Delta p + \frac{\partial y}{\partial q}(t_i, p, q)\, \Delta q$$

Define a vector z with components $y_i - y(t_i, p, q)$, $i = 1, 2, \ldots, N$ and an N by 2 matrix A with components $A(i, 1) = \partial y/\partial p(t_i, p, q)$ and $A(i, 2) = \partial y/\partial q(t_i, p, q)$, $i = 1, 2, \ldots, N$. The payoff at $p + \Delta p$, $q + \Delta q$ is

$$J = \sum_{i=1}^{N} [y_i - y(t_i, p+\Delta p, q+\Delta q)]^2$$

and we wish to choose Δp and Δq so that the new value of J is minimized. If we substitute the approximate value of $y(t_i, p+\Delta p, q+\Delta q)$ into J, we need to choose the vector δ = column $[\Delta p, \Delta q]$ so that the quantity $||A\delta - z||^2$ is minimized. Here $||A\delta - z||^2$ is the sum of squares of the components of the vector $A\delta - z$. This is called a linear least squares problem. The solution is determined by solving

$A^tA\delta = A^t z$ where A^t is A transpose. (There is a better numerical method to determine δ presented in standard linear algebra textbooks.) We change to the new values of p and q and repeat the process several times until either this process converges to good values of p and q or until it is clear the process is not converging. In the latter case, we need new starting values of p and q.

Thus our procedure is to take an initial guess for p and q and solve the initial value problem for w and recording the values of y, u, v at the various t_i measurement points, forming the matrix A and vector z and solving for the incremental vector δ, correcting p and q and cycling thru this sequence of steps until a conclusion arises.

Exercise 4.9: Choose p and q so that the solution of

$$dy/dt = q\, y\, (1 - y^p),\ y(0) = .2$$

best fits the (t, y) data (1, .4), (2, .5), (3, .75), and (4, .9). Thus our criterion for a good fit is to minimize

$$P = [y_1 - y(1, p, q)]^2 + [y_2 - y(2, p, q)]^2 + [y_3 - y(3, p, q)]^2 + [y_4 - y(4, p, q)]^2$$

where $y_1 = .4$, $y_2 = .5$, $y_3 = .75$, $y_4 = .9$ and y(t, p, q) is the solution of the original problem. Choose starting values of p and q near 1 and 1.

Appendix 1
USER Menu Housekeeping

The term user memory refers to that part of the calculator's memory which is accessible to a user through the **VAR** menu on the 48S and the **USER** menu on the 28S. User memory is where we store the various types of objects recognized by the calculator, e.g., real or complex numbers, arrays, programs, lists, etc. These objects are stored as *global variables* (in calculator terminology) which you may regard as the name of the object. Here we are concerned with the basic "housekeeping" procedures associated with programs. By "housekeeping", we mean the simple procedures used to enter, name and store, run, edit and purge programs. The Owner's Manuals minimally address programming; but anyone desiring to become really proficient in developing and using programs across a broad spectrum of applications is strongly advised to study the books "HP-48 Insights" and "HP-28 Insights", by William C. Wickes.

What is an HP-48S/28S program ? A program is a sequence of data objects, procedures, commands and program structures - the *program body* - enclosed between program delimiters:

« *program body* » .

Entering programs. Programs are keyed into the command line and entered onto the stack (level 1) with **ENTER**. You need not key in the necessary closing program delimiters because pressing **ENTER** will automatically insert them for you.

Naming and storing programs. To name and store a program which has been entered onto level 1 of the stack, press ' to signify algebraic entry mode (suitable for entering names and expressions), then key in the desired name and press

[STO]. The program will be stored in user memory under its name, and pressing [VAR] on the 48S (or [USER] on the 28S) will show a user menu key with an *abbreviated* name (up to 5 characters).

To run a program. To run a program, simply press the white menu key beneath the program's abbreviated name; alternatively, key the *full* name into the command line and press [ENTER] [EVAL]. If the program happens to be on level 1, you may simply press [EVAL] . Of course, if the program requires input data for its proper execution then you must first provide that data in an appropriate way, either on the stack or as stored variables which are named in the program body.

Example. The program **« DUP 2 - NEG * »** takes a number "y" from level 1 as input data and returns the calculated value of y(2-y) to level 1. Key in the program by first pressing [« »] on the 48S or [«] on the 28S, then [↰] [ENTER] on the 48S (or the **STACK** menu key [DUP]M on the 28S) followed by the other indicated keys. Press [ENTER] to add the closing program delimiters and copy the program to the stack.

Press ['] **PGM1** [STO] to name this program PGM1 and store it in user memory under this label. Press [VAR] or [USER] to see the menu key [PGM1]M .

Now, run the program using as input data the number 4: key in 4 and press [PGM1]M. The answer, 8, will be displayed on level 1. Notice that you did not have to enter the data onto the stack before pressing [PGM1]M. This is typical; pressing the menu key [PGM1]M automatically entered the data for you. Run the program with some more inputs.

Syntax Errors. When keying a program into the command line, if an object is accidentally entered in an invalid form, then pressing [ENTER] will cause the

calculator to refuse to copy the program onto the stack and display a message indicating a syntax error. To remove the message from the screen so you can correct the syntax, simply press [ON] on the 48S (or [INS] on the 28S).

Example. Key in : « → **ARRY** [ENTER] . Notice what happens. Now remove the message, delete the space after the → and press [ENTER].

Editing programs. To make any change in the body of an existing program you must *edit* the program.

- If the program is on stack level 1, the [EDIT] key will copy it into the command line where you can then make the required changes. Press [ENTER] to return the corrected version to level 1.
- If the program is not on stack level 1, but stored in user memory under, say, 'NAME' the keystrokes ['] [NAME] M [RCL] will recall the program to level 1 and you can proceed as above. Alternatively (and indeed, *preferably*), ['] [NAME] M [VISIT] will copy the program directly from user memory onto the command line for editing; [ENTER] will then replace the old stored program with the newly edited version in user memory.

Example. Start this example with the program « **1 N START NEXT** » stored in user memory as [TRY1] M.

(i) Recall it to stack level 1 with ['] [TRY1] M [RCL] .

(ii) Copy it to the command line with [EDIT], and change **EULER** to **IULER**.

(iii) Copy back to level 1 with [ENTER], then replace the old version by pressing ['] [TRY1] M [STO] .

(iv) Now copy this new version directly to the command line with ['] [TRY1] M [VISIT] , and change **IULER** to **EULER**.

(v) Replace the earlier version by pressing [ENTER] .

(vi) Finally, check your last work by recalling to level 1, examining the result, then dropping it from the stack with [DROP].

HP-48S Shortcuts:

- You can recall to level 1 the contents of any stored variable, say **TRY1**, by pressing [↱] [TRY1]M. Thus, *rightshift will recall.*
- Likewise, you can store (or load) an object on level 1 into any stored variable, by pressing [↰] , then the variable's menu key. Thus, *leftshift will load.* Try this by loading « + SQ COS » into **TRY1**; now recall the contents.

Purging. Imagine that you have stored an object under variable **PGM1** in your user memory. The object may be any one of the variety of objects recognized by the calculator: a real number, an array, a program, etc. To *purge* variable **PGM1** is to remove it and its contents entirely from user memory. Purging a single variable is usually done with the keystrokes ['] [PGM1]M [PURGE]. The label disappears from the menu and its contents are removed from user memory. To purge several variables at the same time press [{ }] on the 48S, or [α] [{] on the 28S, then the menu key for each variable you wish to purge, then [ENTER]. [PURGE] to purge the variables in this list.

Example. Start by storing the numbers 1, 2 and 3 in variables **'X'** **'Y'** and **'Z'** in user memory. With your **VAR** or **USER** menu active, purge **'X'** by pressing 'X left shift [PURGE]; watch [X]M disappear. Now purge the two remaining variables at the same time by building a list { Y Z } and pressing [PURGE]. Watch these variables disappear.

Exercise. The following program takes numbers x, y from the stack and returns sin (xy).

« * **SIN** »

(a) Key in this program and store it under variable "**EX.1**".

(b) Run the program with inputs .5, π. You will get an expression instead of a number. To get a numerical value use the →**NUM** key.

(c) Change the program body by adding **NEG** at the end.

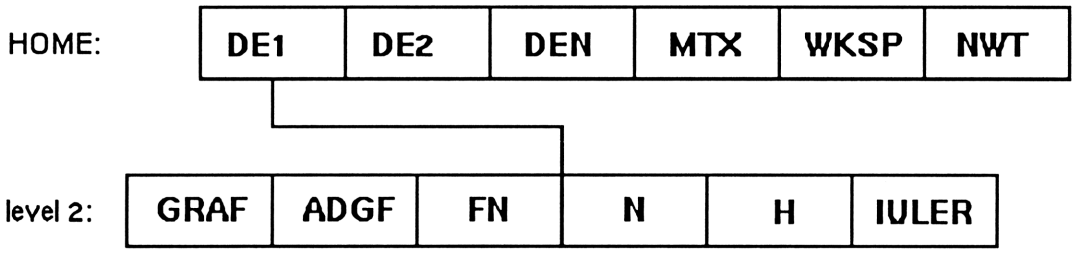
(d) Run the new program with .5, π →**NUM**. What does the new program calculate ?

(e) Purge programs **TRY1**, **PGM1**, and **EX.1**.

Appendix 2
USER Menu Organization: Directories

Just as a file cabinet organizes stored material in an office into convenient groupings, HP-28/48S *directories* enable you to organize the variables and programs that you store in memory. In the HP-28S this memory is called **USER**: in the HP-48S this memory is called **VAR**. The **USER/VAR** memory is itself a directory - the **HOME** directory, and you can always go to **HOME**. Moreover, in much the same way that certain drawers in a file cabinet are further subdivided into sections, you can create *subdirectories* within the directories. The basic idea is to group together variables associated with a particular topic or subtopic.

A convenient directory structure for the material in this book is as follows:



(The variables **EULER PPAR** and **QUIT** should be stored on the next "page", with other programs such as **STP1, RK3** and **ARK. 1** if they are needed.)

HOME contains various entries, one being the **DE1** subdirectory - in which you may group together all the stored quantities for first order differential equations. **HOME** is the *parent* directory of subdirectory **DE1**. The figure shows a directory appropriate for the HP-28S. Programs **ADGF** and **QUIT** are not needed for the HP-48S. The reader should note that programs for the HP-48S in this book have been

structured into directories and loaded into a computer disk which is available from the publisher to instructors who wish to adopt the workbook for classroom use.

Here's how to create our first subdirectory, subdirectory, **DE1**: press ' **DE1** **CRDIR** M (note that **CRDIR** M appears on the **MEMORY** menu). A new label **DE1** M appears in the original **USER/VAR** menu. Pressing **DE1** M will send you to this new **DE1** subdirectory, which is now empty.

On the HP-28S the first variable to store in the **DE1** subdirectory, indeed in *any* subdirectory, is **QUIT** - which will send you back to **HOME**. Create variable **QUIT** by pressing « **HOME** **ENTER** ' **QUIT** **STO** .

You should now transfer the following programs to **DE1**: **EULER, IULER, GRAF**(and **ADGF** on the HP-28S).. Later you will create an **FN** program, and variables **H**, and **N**. Since these programs are likely to be in your **HOME** directory, you will need to transfer them to **DE1**. To transfer a variable from **HOME**, first recall its contents to the stack with **RCL**, activate the **DE1** subdirectory and store the variable name there, then purge it from **HOME**.

You may want to arrange the programs in a particular order in **DE1**. To do this enter { } on the stack. If **GRAF** is to be first (i. e. on the left side of the menu), press ' **GRAF** **ENTER** + to create the list **{ GRAF }**. If the next program is to be **ADGF**, press ' **ADGF** **ENTER** + to create **{ GRAF ADGF }**. Continue in this way until you have an ordered list of the elements of DE1 that you care to order. Then press **ORDER** on the **MEMORY** menu to rearrange the menu.

Subdirectory **DE2** should contain **GRXY, GRXT, ADG.2, FN, GN, N, H, IULE2, EULE2,** and **PPAR. DEN** should contain **IULN, EULN, RK3A,** and **STPN. NWT** should contain the programs for Newton's method to find the solution of several equations. **MTX** should contain the **PIV, ROKL,** and **CHAR** programs from Chapter 4. You may want an **WKSP** directory for odds and ends. The serious differential equations student should create a subdirectory for the programs in Appendix 4.

Appendix 3

Starting Points for Newton's Method

This appendix gives an outline of programs to determine an approximate location of the solution of two equations f(x,y) = g(x,y) = 0 in two unknowns. We select a rectangle in the x,y plane, a grid within that rectangle, evaluate a function, say f(x,y) at the grid points and turn on a pixel at the grid point if the function is positive at the point. The boundary of the points with pixels showing is the set of points with value 0. We select several points we think are on this boundary using the **INS** key and connect the points with straight lines using the lines program. We repeat for the other function and superimpose the graphs containing the line segments. Intersections are starting points for Newton's method for the solution of a set of equations:

$$w_{k+1} = w_k - J^{-1}(w_k)\begin{bmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{bmatrix}, \text{ where } w = \begin{bmatrix} x \\ y \end{bmatrix}.$$

We select the rectangle by choosing plot parameters to show $x_m \le x \le x_M$, and $y_m \le y \le y_M$, and the grid by

$$H1 = (x_M - x_m)/N,\ K1 = (y_M - y_m)/P$$

$$x_i = x_m + iH1,\ y_j = y_m + jK1,\ i = 0, 1, \ldots N,\ j = 0, 1, \ldots P$$

The subprogram **STG1** sets the grid size. The input to **STG1** is a pair of positive integers (for N, P).

Program Name	**STG1**
Purpose	Sets the grid size, evaluates f at grid and plots those points where f > 0
Stored Quantities	**STG2 STG3 FN** plot parameters

Input		Output
level 2	level 1	
N	**P**	graph with active cursor

```
<< 'N' STO 'P' STO PPAR 1 GET DUP PPAR 2 GET
SWAP - C→R P / 'K1' STO N / 'H1' STO C→R
STG2 'K1' PURGE 'H1' PURGE >>
```

The second subprogram **STG2** evaluates a function at the grid points. The

Program Name	**STG2** HP-48S version
Purpose	Evaluates f at grid and plots those points where f > 0
Stored Quantities	**PPAR STG3 FN**

Input		Output
level 2	level 1	
x_m	y_m	graph with active cursor

```
<< { # 0d # 0d ) PVIEW K1 + 1 P 1 - START 1 N 1 -
START SWAP H1 + SWAP DUP2 STG3 NEXT K1 + SWAP
H1 N 1 * SWAP NEXT GRAPH >>
```

Because the program for double loop has not arisen in these notes we give here a stack accounting at various stages in the program.

<< { # 0d # 0d } PVIEW	
K1 +	at this point the stack is x_m y_1
1 P 1 - START	call this stack x_0 y_j
1 N 1 - START	call this stack x_{i-1} y_j then
SWAP H1 + SWAP	x_i y_j
DUP2 STG3 NEXT	bottom of loop on i
K1 + SWAP H1 N 1 - * - SWAP	now stack becomes x_0 y_{j+1}
NEXT GRAPH >>	bottom of loop on j

For the HP-28S replace **{ # 0d # 0d } PVIEW** with **CLLCD** and omit **GRAPH** in the programs above.

The purpose of the following subprogram **STG3** is to test if f(x,y) > 0, if so record a pixel, if not continue. The input is a pair of numbers x y. For the HP-48S

<< DUP2 FN 0 IF > THEN DUP2 R→C PIXON >>

(replace **PIXON** with **PIXEL** for the HP-28S). Finally **FN** is a subprogram to calculate the value of a function of x and y whose 0 level curve is to be studied. It should take x y off the stack and produce f(x, y).

When the program has been executed select several points on the boundary of the plotted points and connect them with lines. (On the HP-48S, this can be done using the cursor and the **LINES** command. On the HP-28S select the points by using the **INS** key. When the stack display has been activated compile these points into a list which is stored as **L1**. This list can be used as input for the program **SKETCH** given in the book *HP-28 Insights* by William C. Wickes, Larken Publications 4517 NW Queens Avenue, Corvallis, Oregon 97330 (notice a program called **LINES** must also be entered).) In both cases we have produced a crude plot of the approximate

zero level curve of f. Store the graph as a string. Repeat this for the function g(x,y) (this time storing the list as **L2**) and obtain a plot of the approximate zero level curve for g on the same graph. Use the cursor to indicate any intersection. Such an intersection is a starting point for Newton's method.

Appendix 4
Runge-Kutta Adaptive Step Size Algorithm

Elementary algorithms have been featured in this workbook primarily for pedagogical reasons; however they also permit quick execution times. There is a loss of accuracy, but it is thought the simplicity of the programs will permit students to learn valuable programming skills. In this appendix, we present an extension of the adaptive step method presented in the first chapter for the solution of initial value problems. The extension is a program for the popular Runge-Kutta Feldberg 4/5 algorithm which calculates the new value of y using a result which is accurate to fifth order in h whenever an estimated error test is passed. When the test is successful the step size is increased and the next step is attempted. If the test fails, then the step size is reduced and another attempt is made to find a step size in t which gives at most an extremely small error in the new value of y. For simplicity, in our programs no check is made to prevent extremely small steps. If the user wants to prevent ridiculously small steps, appropriate program steps should be incorporated. The algorithm is given by:

$$y_{new} = y + (16/135)k_1 + (6656/12825)k_3 + (28561/56430)k_4 - (9/50)k_5 + (2/55)k_6$$

and

$$esterr = (1/360)k_1 - (128/4275)k_3 + -(2197/75240)k_4 + (1/50)k_5 + 2/55\,k_6$$

where

$$k_1 = hf(t,y),\quad k_2 = hf(t+h/4,\ y+k_1/4)\quad k_3 = hf(t+3h/8,\ y+3k_1/32+9k_2/32)$$

$$k_4 = hf(t+12h/13,\ y+1932k_1/2197-7200k_2/2197+7296k_3/2197)$$

$$k_5 = hf(t+h,\ y+439k_1/216 - 8k_2+ 3680k_3/513 - 845k_4/4104)$$

$$k_6 = hf(t+h/2,\ y - 8k_1/27 + 2k_2 - 3544k_3/2565 + 1859k_4/4104 - 11k_5/40)$$

Given t, y, and h, the process of calculating the term y_{new} using these formulae is coded into the following program. At the end of the program, the step size is divided by 2 and stored.

Subprogram Name	**FBG**
Stored Quantities	**FN H**
Input	**T** and **Y**
Output	**T Y esterr** (**H** has been reduced 50% & **DELY** has been stored)

```
<< → T Y
<< T Y FN H * DUP DUP2 3 DUPN 4 / Y + T H 4 / +
SWAP FN H * DUP DUP2 9 * 5 ROLL 3 * + 32 / Y + T H
3 * 8 / + SWAP FN H * DUP DUP DUP2 7296 * 6 ROLL
7200 * - 8 ROLL 1932 * + 2197 / Y + T H 12 * 13 / +
SWAP FN H * DUP DUP2 845 * 4104 / 5 ROLL 3680 * 513
/ SWAP - 8 ROLL 8 * - 9 ROLL 439 * 216 / + Y + T H +
SWAP FN H * DUP DUP 11 * 40 / 4 ROLL 1859 * 4104 /
SWAP - 6 ROLL 3544 * 2565 / - 8 ROLL 2 * + 8 ROLL 8 *
27 / - Y + T H 2 / + SWAP FN H * DUP 2 * 55 / 3 ROLL
9 * 50 / - 4 ROLL 28561 * 56430 / + 5 ROLL 6656 * 12825
/ + 6 ROLL 16 * 135 / + 'DELY' STO 2 * 55 / SWAP 50 /
+ SWAP 2197 * 75240 / - SWAP 128 * 4275 / - SWAP 360
/ + ABS T Y ROT H 2 / 'H' STO END >> >>
```

The reader should notice that to calculate y_{new} and esterr, k_1 is needed 7 times (once each for k2, k3, k4, k5, k6 y_{new} and esterr), k_2 is needed 4 times (once each for k_3, k_4, k_5, k_6), k_3 is needed 5 times (once each for k_4, k_5, k_6, y_{new}, esterr), k_4 is

needed 4 times (once each for k_5, k_6, y_{new}, esterr), k_5 is needed 3 times (once each for k_6, y_{new}, esterr), and k_6 is needed 2 times (once for y_{new} , once for esterr). In the program listing, each time the commands FN H * are executed, one of the k terms has been calculated. The next commands create the appropriate number of copies of these numbers.

The **FBG** program is used as a subprogram to the program **RKFS** program listed below:

Subprogram Name	**RKFS**
Stored Quantities	**FN H FBG**
Input	**T** and **Y**
Output	T_{new} Y_{new} **H** (**H** has been altered)

```
<< DO FBG UNTIL .00001 < END DELY + SWAP H 2 * +
                SWAP H 4 * 'H' DTO    >>
```

The tolerance used in the program (.00001) may require adjustment. In fact, this quantity can easily be changed to a variable and used as input to the driver program listed below:

Program Name	**RKF**
Stored Quantities	**FN H FBG RKFS**
Input: $T_{initial}$ $Y_{initial}$ T_{final}	Output: **Stored List YV**

```
<< → TF << { } 'YV' STO DO RKFS DUP2 OBJ→ DROP
 2 →ARRY YV + 'YV' STO 2 PICK H + UNTIL TF > 2
PICK TF SWAP - 'H' STO FBG DROP DELY + SWAP 2 H *
  + OBJ→ DROP 2 →ARRY YV + 'YV' STO   >> >>
```

As written, the program is designed for a single differential equation. However, the entire program will work for a vector system dy/dt = f(t,y) with y and f as vectors with M components except the user should replace in two places in **RKF** the sequence of commands **OBJ→ DROP 2 →ARRY** with **OBJ→ DROP** (M+1) **→ARRY** Here substitute the number M+1 into the command or use a variable named **M** and put **M 1 +** in for (M+1). Of course, the program **FN** takes **T** and the vector **Y** from the stack and returns the vector f (**T**,**Y**).

Appendix 5
Graphs for an Input-Output System

We return here to an important linear second order differential equation with constant coefficients, a forcing term f(t) and zero initial conditions. The forcing term can be regarded as an input to the system and the resulting solution as the corresponding output. Our subject will again be a spring model

$$\frac{d^2y}{dt^2} + 2b\frac{dy}{dt} + \omega^2 y = f(t), \quad y(0) = \frac{dy}{dt}(0) = 0, \; \omega^2 > b^2 .$$

The solution is given by

$$y(t) = \frac{1}{\sqrt{\omega^2 - b^2}} \int_0^t e^{-b(t-s)} \sin\sqrt{\omega^2 - b^2}\,(t-s)\, f(s)\, ds.$$

The problem here is to graph the output function y(t) for a given input function f(t) using the integration program provided on the calculator. In Chapter 3 we did this problem using **IULE2** and **GRXT** partly because the student should become thoroughly familiar with those important programs. Now we want to use the calculator integration program to do the same problem because our treatment for this problem is an excellent exercise on the basic properties of exponential functions evaluated at suitable complex numbers.

We want to calculate and graph the output function at points $\{(t_j , y_j)\}$ for an input function suitably stored where $t_j = jH$ for $j = 0, 1, 2, \ldots N$ and we want to

avoid recalculating any parts of the integral already computed. To do this we will take advantage of a key property of the exponential function for complex argument, namely that the product of two such expressions can be obtained by adding the exponents. Consequently the program involves complex arithmetic. First we define

$$z(t_j) = \int_0^{t_j} e^{(-b+i\mu)(t_j-s)} f(s)\, ds, \quad j = 1, 2, \ldots N$$

where $\mu^2 = \omega^2 - b^2$ and after some work we note

$$z(t_j) = e^{(-b+i\mu)h}\, z(t_{j-1}) + e^{(-b+i\mu)(t_{j-1}+h)} \int_{t_{j-1}}^{t_{j-1}+h} e^{-(-b+i\mu)s} f(s)\, ds$$

(The student is urged to verify this equation!) Even though $z(t_j)$ is an integral from 0 to t_j, to calculate $z(t_j)$ we need only $z(t_{j-1})$ and an integral on the interval t_{j-1} to t_j !

We want to calculate only enough point to recognize the output graph. That is, N does not need to be large since there is no loss of accuracy for small N. Suppose we store the numbers μ, b, and h in U, B and H and we create and store the following subprograms for the integrands

INT1 'EXP(B*X*)*COS*(U*X)*F(X)' **INT2 'EXP(B*X)*SIN(U*X)*F(X)'.**

Subprogram Name	**SP.3** HP-48S version
Input	t
Output	$\left(\int_t^{t+H} e^{bs} \cos \mu s\, f(s)\, ds,\ -\int_t^{t+H} e^{bs} \sin \mu s\, f(s)\, ds \right)$
Stored Quantities	**H INT1 INT2**

```
<< 'X' PURGE 3 FIX DUP DUP H + DUP 3 ROLLD INT1 X 3
   NEG SF ∫ 3 ROLLD INT2 X ∫ 3 NEG CF
   NEG R→C >>.
```

The HP-28S version is given by

SP.3 for the HP-28S (same input, output, stored quantities)

```
<< 'X' PURGE X SWAP DUP H + 3 →LIST DUP INT1
SWAP .001 ∫ DROP SWAP INT2 SWAP .001 ∫ DROP
NEG R→C >>
```

Subprogram Name	**SP.1**
Input	z(t) t
Output	z(t+h) t+h
Stored Quantities	**SP.3 H B U**

```
<< DUP DUP SP.3 SWAP H + B NEG U R→C * EXP *
  3 ROLL B NEG U R→C H * EXP * + SWAP H + >>
```

Subprogram Name	**SP.2**
Input	z(t) t
Output	z(t) t and a point (t, z(t)) on the graphing screen:

<< DUP2 SWAP C→R SWAP DROP R→C PIXON >>.

(For the HP-28S replace **PIXON** with **PIXEL**)

Next we set the plot parameters for the output graph and execute a program loop to calculate and plot the output graph.

Program	**GR.Y**	HP-48S version
Purpose	Graph output of the second order IVP	
Stored Quantities	**SP.1 SP.2 SP.3 N XRNG YRNG**	

<< { # 0d # 0d } PVIEW DRAX (0,0) 0 SP.2 1 N START SP.1 SP.2 NEXT GRAPH >>.

In the HP-28S program, replace **{ # 0d # 0d } PVIEW** with **CLLCD DRAX**, replace **GRAPH** with **DGTIZ LCD→** store **PMIN, PMAX** instead of **XRNG YRNG**.

Problem: Find the output graph for $f(x) = 1 - \sin^4(4x)$ for $\mu = 1$, $b = .5$, $N = 25$ and $H = 3.14/N$. (By the methods in Chapter 2, the output function for this input function can be obtained from a table of integrals after several substitutions. An output function for an input function such as $f(x) = 1/(1 - \sin^4(3x))$ could not.)

Program Index

First Order Initial Value Problems

EULER, IULER Euler, improved Euler algorithm 3-4
GRAF Graphics program (uses **IULER**) 6
ADGF Program for adding a new graph (HP-28S) 7
IESV1 Improved Euler algorithm: creates solution value list 12
STP1 Adaptive step size algorithm: single step 19
ARK.1 Adaptive algorithms: initial point to terminal point 22
DDS Discrete dynamical system 24
INIT1 Initialization program for **GRAF** 26

Initial Value Problem Variables and Parameters

DER et al. Programs for parameter identification problem 42-44
DLAY Program for delay differential equation system 52

Initial Value Problems: Two Differential Equations

EULE2, IULE2 Euler, improved Euler algorithm: 2 differential equations .. 57-58
GRXY, GRXT Graphics: solutions of 2 differential equations 60
NSTP et al Program for Newton's method: simultaneous equations 75
STPN Variable step algorithm for vector autonomous system 79

Higher Order Systems

EIG3 Eigenvalue equation for 3 by 3 matrices 86
PIV Pivot routine for n by n matrices 88
ROKL Interchange rows K and L 88
CHAR Characteristic equation for n by n matrix 93
EULN, IULN Euler, improved Euler algorithm for vector system 95
RKF Runge-Kutta Feldberg 119

Subject Index

Acoustic 79
Asymptotic vii, 1, 23, 62, 69, 82
Attracting solutions 8, 67-8, 74, 98
Autonomous 67
Characteristic equation 32, 72, 81, 86
Characteristic equation algorithm 93
Critical point 67, 74-5, 85
Delay problems 51
Discrete dynamical system 23, 82
Eigenvalues, eigenvectors 72, 85-94, 97
Euler algorithm 2, 56, 95
Falling body problem 10, 31
Gauss-Jordon echelon form 88
Graphics screen 6
Inflection points 10
Improved Euler algorithm 4, 56, 95
Input signal delay 14
Inverse function problem 28
Linear autonomous system 71
Lorentz equations 97
Lotka-Volterra model 67
Mixing problems 13, 29, 49
Newton's law 10
Newton's method 41, 74, 113
Numerical integration 12, 49, 63, 121
OR command, use to combine screens 7
Orthogonal trajectories v, 65
Overdrawing 7
Parameter identification 38, 103
Pendulum 68
Periodic response vi, 48, 62, 67
Phase plane 67
Population growth equations 9, 54
Pursuit 64, 95
Ramp function 15
Recursive function 23, 82
Repelling solutions 67, 98
Restricted three body problem 100
Runge Kutta algorithm 18-20, 76, 117
Saddle point 73
Satellite motion 100
Second order initial value problem 56
Sliding bead 32
SOLVE 29
Spiral point 72-3
Spring motion 46, 61, 121
Square wave 17
Step size selection 5, 18, 76, 117
Storage of results in a list 11
Switch function 17
Taylor's theorem 18, 40, 76
Vander Pol 69
Variational matrix 74, 97
Water storage problem 11

Also available from Saunders College Publishing in the Clemson Series:

- **Calculator Enhancement for Introductory Statistics**
  By I.B. Fetta
- **Calculator Enhancement for Linear Algebra**
  By D.R. LaTorre
- **Calculator Enhancement for Single Variable Calculus**
  By J.H. Nicholson
- **Calculator Enhancement for a Course in Multivariable Calculus**
  By J.A. Reneke
- **Calculator Enhancement for Precalculus**
  By I.B. Fetta

D.R. LaTorre, Clemson University, Consulting Editor

ISBN 0-03-092730-7

9 780030 927300 90000