Introduction to MLAB
The Royal Road to Dynamical Simulations

Revision Date: September 24, 2015

Foster Morrison, author
Turtle Hollow Associates, Inc.
P.O. Box 3639
Gaithersburg, MD 20885 USA

Daniel Kerner, editor
Civilized Software, Inc.
12109 Heritage Park Circle
Silver Spring, MD 20906 USA
Email: csi@civilized.com
Web-site: http://www.civilized.com
(301) 962-3711

Contents

PREFACE

The famous mathematician Euclid reportedly told the ruler of Egypt, Ptolemy I, "There is no royal road to geometry." Even the powerful and mighty would have to struggle through the thirteen parts of Elements to master the subject.

Geometry cannot satisfy all the needs of contemporary scientists and engineers. The subject is highly abstract and wonderfully rigorous, but far too constrained to model things that are intricate or dynamic. We need differential equations, statistics, a few other odds and ends, and computers.

Most technical people are not mathematicians or programmers and they do need a "royal road" to dynamical simulations and the other modeling chores associated with their work. Being a chemist, biologist, engineer, or other specialist is just as demanding in its own way as serving as head of state, so it is not also feasible to be an expert in applied mathematics and computer science.

Those with considerable needs and matching resources can hire specialized staff members. For everybody else there are now available a number of computer programs that facilitate various modeling tasks. One that is highly versatile, and the subject of this introductory manual, is `MLAB` (for `M`athematical `LAB`oratory).

MLAB was initiated at the National Institutes of Health in the 1970s to enable a wide variety of researchers to create essential mathematical models. The program is an interpreter, which means it executes commands as they are received, like the GW-Basic provided with many personal computers. Where MLAB differs from Basic is in being less general purpose and more focused on performing certain critical modeling computations without needing to code them on a step-by-step basis.

Many software packages can integrate ordinary differential equations numerically, but MLAB is one of the very few that also can adjust parameters and initial conditions. MLAB is equally facile at handling curve fitting, where the adjustments are linear or, as is the case with most differential equation models, nonlinear. MLAB is a convenient calculator and handbook of functions, as well as a programming language, and it has a publication quality graphics capability.

This user's guide serves two purposes. First it will acquaint you with the MLAB syntax, which resembles Basic, Pascal, C, and their progenitor, Fortran. Some knowledge of any of these will

make MLAB immediately familiar, at least up to a point. While you are acquiring MLAB skills, the examples will also illustrate some of the important principles of pure and applied mathematics, as well as modeling and simulation.

Read the text and then execute the *.DO file demonstrations supplied for each chapter that appear in Appendix B. Be sure to study the insertedc omments that flash on the screen, as well as the active MLAB commands. (The *.DO files are nothing more than ASCII code and can be printed for easy reading; a better idea is to use your word processor, since some of the lines may be longer than your paper. You can continue to review MLAB and expand your understanding of modeling by modifying the supplied *.DO files. Start with the suggestions in Chapter 10 and then let your imagination run.

This manual is designed to introduce modeling and MLAB usage in the simplest possible ways, rather than the most general. Some carefully selected, readily available modeling texts are listed in a brief bibliography; working through this manual first will make these books more accessible.

Ultimately, the greatest contribution of MLAB will be to education. Every citizen must become aware of dynamical concepts and principles, not just the static precepts of geometry. Change, equilibrium, stability, and random walks should be familiar to everybody, not just the experts in modeling and mathematics. MLAB can be used to demonstrate these concepts to the vast majority who will not specialize in mathematics or other technical disciplines. For those who do pursue careers in science and engineering, a moderately sophisticated appreciation of dynamics and modeling is essential.

Details on all the capabilities and syntax are given in the **MLAB Reference Manual**. An advanced course, weighted in favor of life sciences and chemistry, is provided in the **MLAB Applications Manual**. The total collection of *.DO files on the disk provides paradigms of numerous modeling problems. In many cases these can be adapted to your own needs with minimal modifications.

A primary purpose of this introduction is to help potential users decide to purchase MLAB. Working through this manual with the trial version of MLAB will enable you to determine how much of your work can be done with MLAB. For some specialists the answer is “none” and that will be obvious; but for many others the portion will be 100 per cent.

When MLAB is applicable to your work, its value may be substantial. What would take days or even months to do with programming can be done in a half an hour in MLAB.

More than just useful, modeling is an enjoyable activity. The symmetries, or lack thereof, of mathematical constructions have certain aesthetic values; a few are rather entertaining. Now let MLAB and its graphics introduce you to a few such examples.

Thanks are due to Marvin Stein for introducing me to MLAB and Gary Knott for much instruction on its syntactical subtleties. My wife Nancy provided support by editing the manuscript and producing the final camera ready copy.

The MLAB programs (*.DO files) were developed and tested on a Heath-Zenith H-241 personal computer with an Intel 80286 processor and 80287 floating point coprocessor and an EGA monitor. The initial version was produced on this same computer with Microsoft Word, Version 4.0.

Foster Morrison
North Potomac, MD 20878
1992 November 10

The original 1992 manuscript in Microsoft Word format was updated and translated to the LaTeX markup language. The resulting manuscript was then processed with the *latex* program to generate a Device Independent (.dvi) file; the resulting dvi-file was then converted to a PostScript (.ps) file using the *dvips* program. The resulting ps-file was then converted to Portable Document Format (.pdf) format with the *ps2pdf12* program. The *latex*, *dvips*, and *ps2pdf12* programs are standard tools included with Linux computers.

The MLAB do-files and their graphical output appearing in Appendix B were tested on personal computers with Microsoft Windows XP and 7, Kubuntu Linux, and Apple Macintosh OSX operating systems.

Chapter names in the Table of Contents, figure numbers in the text and List of Figures, section numbers of do-files in Appendix B, and page numbers in the index, appear in the color blue and provide clickable links to the respective content.

Daniel Kerner
Silver Spring, MD 20906
2014 February 28

Chapter 1

BASIC MATHEMATICS AND ITS IMPLEMENTATION IN MLAB

The MLAB Interpreter is designed to make calculations, especially the modeling of dynamic systems, as easy as possible for the expert, the student, and all those whose special expertise is not applied mathematics. This chapter will introduce the basic syntax of MLAB by demonstrating a few calculations and how they may diverge from what mathematics predicts.

Subsequent chapters introduce modeling of a modest degree of complexity and the principal MLAB functions provided to do it, as well as a few of the specialized functions. The most important ideas of modeling, error control and system resolution, are woven into the examples. These can be accomplished readily with MLAB by those without extensive training in mathematics and computer science.

1.1 What You Need to Know

MLAB users should be familiar with the basics of their computer and its operating system. This manual and its examples were created on an MS/DOS IBM-AT compatible. MLAB is also available for Linux with X-Windows, Microsoft Windows, and Macintosh systems. The more recent systems will be less constrained by available memory. Since MLAB is controlled by command line inputs and does not have pull-down menus, the user needs a corresponding degree of computer literacy.

High school algebra and a little trigonometry are the mathematical minima. Some intuitive understanding of calculus and differential equations will make the demonstrations easier to comprehend. There is no need for refined skills in mathematical manipulation or to review those long forgotten theorems. The examples used in this manual do not require sophistication in statistics.

Matrices are rectangular arrays of numbers; vectors are the special case of a single column. The properties of vectors and matrices used in the demonstrations are described in the text and comment statements. Some mention is made of complex numbers.

It is easy to read the text and run the demonstrations. As you need to know more details about MLAB, consult the **MLAB Reference Manual** and the **MLAB 2-D Graphics Examples**. Details of syntax are spelled out in the **MLAB Reference Manual**. Included in Chapter 10 is a reading program on modeling that can be completed in a few weeks rather than a few years.

1.2 Getting Started with MLAB

Before installing MLAB, locate a ruler (English or metric) that can measure the height and width of your computer screen. This will be needed to calibrate the graphics output. Check that your computer meets the minimum specifications for installing and running MLAB and (assuming all is well) just follow the instructions.

But first read Section 1.3 and Section 1.4. Then install MLAB, if necessary, follow the directions in Section 1.5, and practice the most basic operations. You will be able to use MLAB to illustrate the principal concepts of nonlinear dynamics and the failure of computations to agree with mathematics.

1.3 Foundations of Mathematics

Not all scientists and engineers are enthralled with mathematics. For one thing, the subject is presented in too formal a manner, in the guise of rigor. But mathematics, like the sciences and technologies, has changed over the centuries and over the past few decades. In fact there are now several schools of mathematics vying for the mainstream.

More than a few people, both experts and laymen, have observed that logic is not very logical. The axiom that asserts either a statement or its negation must be true does not hold strictly for the vague constructions of natural languages, such as American English. Even in your computer memory the alternatives are "true" (1), "false" (0), and "parity error." Reasoning that holds for a finite, failure-free system is less than satisfactory for the complexities of the real world or even the open-ended "infinite" constructions of pure mathematics.

For this presentation we have chosen as a formal basis the constructive analysis pioneered by the late Errett Bishop (1928-1983). For one thing its assumptions for mathematical existence coincide with any possible computer implementation, but are not constrained by technical feasibility. Only finite, convergent algorithms are accepted as proof of existence. The infinite searches of vaguely

defined sets that characterize much of recent mathematical research are deemed "interesting," but in need of further work.

In doing computations we have to work with a depleted set of rational numbers; the so-called real numbers are available only when we engage in the practice of mathematical analysis. There are a number of software packages that can do "formal operations"; in MLAB this capability is limited to differentiation of functions, illustrated in the chapter on Hamiltonian methods. The more that can be done formally, the better off will be the modeler in every way.

Constructive analysis differs from numerical analysis in that it deals with computational errors by establishing rigorous upper bounds; numerical analysis draws heavily on linearizations and statistical error estimates. From a scientific standpoint, this surely is the sensible way to draw the line between pure and applied mathematics.

1.4 Sets

Georg Cantor (1845-1918) tried to establish the notion of "belongs to" as the most basic element of mathematics. This was only the latest skirmish in the long war between geometry and arithmetic for dominance in mathematical thinking. Geometry, which had been dominant in Europe since the Graeco-Roman era, had been losing ground to arithmetic and algebra since the invention of calculus in the 17th century.

The theory of sets had a certain appeal. Numbers, as well as geometric concepts, could be expressed in terms of various sets. There were sets of numbers for algebra and calculus, sets of open sets for the generalized geometry called topology, and sets of functions for the abstract "spaces" of advanced analysis. There were also sets of oranges and apples. Indeed, sets encompassed all possible combinations of all things, abstract and material.

Such a generalized concept for sets led to many paradoxes and other logical problems which were patched up in various ways. From a scientific viewpoint, however, there are serious objections to making classification a part of mathematics. When an object leaves the set of chairs and enters the set of firewood may not be crisply defined. And from a mathematical viewpoint we are faced with an "existence" of abstractions with no computational realization possible.

L.E.J. Brouwer (1881-1966) proposed that all mathematics be based on the arithmetic of the positive integers (1, 2, 3, 4, ...). After all, if a fundamental contradiction were ever found in so simple and basic a concept, it would be time for mathematics and all quantitative analysis to hold a "going out of business" sale. However, Brouwer never developed more than the most basic needs for analysis and most of the mathematical community continued to build their work on the foundations laid down by Cantor.

In constructive analysis a set is considered legitimate only if it describes a mathematical abstrac-

tion, specifically a finite algorithm based on the arithmetic of rational numbers (and ultimately on the positive integers). A set may be somewhat vague, in that it specifies a class of algorithms for constructing any of its members, not necessarily every one. Making just the right distinction between any and every is critical for getting beyond the basics of counting to concepts based on geometric intuition.

Infinite, or more properly "non-terminating," constructions like real numbers have some properties that can be determined and others that can be addressed, but never established. (What are the statistics of the digits in the decimal expansion of π, 3.141592653589 ... ?) Formal analysis pretends to determine some of these properties, but only by a process that can never be implemented.

This brief introduction is meant to explain why we define a few things differently from what may be done in your favorite math text. Books on advanced calculus, differential equations, linear algebra, and applied mathematics almost never address questions sensitive to the differences between constructive and formal analysis in any depth. But many students not seeking a doctorate in pure mathematics are exposed to topology, abstract analysis, and modern algebra. We believe constructive analysis supplies the best answers to mathematical questions of interest to scientists and engineers, rather than leaving them to sort things out by intuition.

Above all, it helps keep clear in ones mind the distinctions among a mathematical model, its computational realization, and the system of interest being observed. For our presentation we have some equations, some MLAB statements, and in some examples, simulated data created by other MLAB statements. Modeling consists of making all these things, and actual observations, share some certain properties to a desired, attainable level of precision. Other properties of these entities may be quite different and therein lies the source of much folly among popularizers and sensation seekers.

1.5 Numbers, Prime, Rational, and Real

Apples and oranges may be sliced, as well as counted, so fractions were invented in prehistoric times. The modern formalism is rational numbers, which are pairs of integers, say (m, n), or more commonly, m/n. First it is a good idea to add zero and the negative integers to the positive whole numbers, even though this is not how things happened historically.

The good news about rational numbers is that they have potentially limitless resolution: 1/10, 1/100, 1/1000, ..., $1/1,000,000$, ... The bad news is that they soon become unworkable. There are an unlimited number of prime numbers, which are those positive integers that can be divided evenly only by 1 and themselves. The first few are $1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29$; larger examples are $1009, 1013, 1019$, and 1021. In grade school children are taught to reduce fractions, as

$$\frac{105}{385} = \frac{[(3)(5)(7)]}{[(3)(5)(11)]} = \frac{7}{11}. \qquad (1-1)$$

However, large integers are not readily factored, even on a computer.

Most calculations of any consequence are done with floating point numbers. What this does is constrain denominators to a few special cases and replaces most rational numbers with a select few that are close in magnitude. For example

$$\frac{7}{11} = \frac{63,636,364}{100,000,000} \tag{1-2}$$

approximately. The standard notation is

$$\frac{7}{11} = 0.63636364 \tag{1-3}$$

Denominators in decimal arithmetic are powers of 10, so the only rational numbers represented exactly are those whose denominators are products of 5s and 2s. On computers denominators are usually restricted to powers of 2 $(1, 2, 4, ..., 1024, ...)$, which really is neither worse nor better mathematically, but makes for more reliability and efficiency. This is truncation error.

More bad news is that not everything of interest can be represented exactly with rational numbers, no matter how many prime factors in the denominator. The square root of 2 was the first known example, but the useful values $\pi = 3.14159...$, $\mathrm{e} = \exp 1 = 2.71828...$, among others, share this unhappy trait. Hence it is necessary to introduce real numbers, which really are well-defined infinite sequences of rational numbers. Discrepancies between real numbers and floating point approximations are another form of truncation error.

Some truncation errors can be avoided by doing formal mathematical analysis. Unfortunately, not all mathematical operations of interest can be accomplished by formal manipulations, such as

$$\sin^2(x) + \cos^2(x) = 1 \tag{1-4}$$

Square roots of sums cannot be simplified, though products are easy. Few differential equations can be solved formally, and of those that can, only the nearly trivial cases are useful.

The arithmetic operations done in a computer only approximate the results specified by mathematics. To agree with theory perfectly would require the use of ever longer representations, so that the entire computer memory would soon be filled with one very precise number. For divisions this can happen in a single step! So floating point arithmetic is executed with a fixed percentage error, at least for multiplication and division. Subtraction is the most dangerous operation. If two numbers are identical in sign and similar in magnitude, their computed difference will have a much higher percentage error than the original operands.

Discrepancies between mathematical arithmetic and floating point arithmetic are called rounding errors. In any computation beyond the most trivial, rounding and truncation errors interact to play havoc with your results. Good algorithms tend to keep the total error to the minimum possible. Computation errors are a kind of background noise. As such, they can be modeled and controlled just like the background noise observed in engineering and scientific problems. The functions provided in MLAB are designed to contain numerical errors, but the user should do simulations to test modeling algorithms. Examples will be given as you progress through this Introduction.

1.6 Try Out MLAB

A compressed file containing the MLAB application, ancillary files, and demonstration-script and data files, can be downloaded from the Civilized Software web-site at `http://www.civilized.com`. Be sure to download the correct compressed file for your computer. Different files are available for Linux with X-Windows, Microsoft Windows, Apple Macintosh computers with Motorola microprocessors, Apple Macintosh computers with PowerPC microprocessors, and Apple Macintosh computers with Intel microprocessors. DOS versions (640K and extended DOS) of MLAB are available on diskettes.

On Microsoft Windows computers, downloading the MLAB package and running the MLAB setup program will create the directory `C:\MLAB`, uncompress the MLAB program and its ancillary files to that directory, and create an MLAB icon on the Desktop. To start MLAB, double-click the MLAB icon on the desktop with the mouse.

On Apple Macintosh OS7, OS8, OS9, and OSX systems, installing MLAB will create an MLAB folder on the desktop and uncompress the MLAB program and its ancillary files to that folder. To start MLAB, open the MLAB folder on the Desktop and double-click the MLAB icon with the mouse. (You may find it convenient to open the MLAB folder, click on the MLAB program icon, create an "Alias" for the MLAB program using the Finder's FILE menu, and drag the resulting MLAB Alias to the desktop. Then you can run MLAB simply by double-clicking on the desktop MLAB Alias.)

On Linux systems with X-Windows, once you download the MLAB package from `www.civilized.com`, you must log-in as administrator and issue a `tar` command. This will create an MLAB directory on `/usr/local/lib/mlab`, unpack the required MLAB files in that directory, and create bash shell scripts to run MLAB in the directory `/usr/local/bin`. You can then start MLAB by opening an XTERM terminal window and typing the command `mlab` (or `MLAB`) after the bash shell prompt. (Depending on the desktop environment running on your Linux system–Kubuntu, Gnome, Kde, etc.–you can create an icon on the desktop which is a shortcut to the MLAB program. Be sure to indicate that the shortcut is to run MLAB in a terminal window, when creating the shortcut. Then you can run MLAB simply by clicking on the desktop icon.)

On DOS computers, MLAB is installed from diskettes. Running the install program on the first diskette will uncompress the MLAB program and ancillary files to the directory `C:\MLAB`. You can then start MLAB by going to the directory where the MLAB executable file is located——using the `cd` command, and typing `mlab`.

Once extracted and installed, start MLAB.

In all cases, after some electronic boilerplate flashes by on the screen, you will see a menu, welcoming you to `MLAB` and offering you three choices:

1. Look at a list of MLAB demonstration do-files.
2. Go to top-level MLAB.
3. Display some tips on using MLAB.

If you are new to `MLAB`, you should select the first option and click the CONTINUE button or strike the Enter/Return key. You will then be presented with a dialog window with instructions for running various canned scripts–called `DO-files`–that demonstrate some of the capabilities of MLAB. After clicking on the CONTINUE button or striking the Return/Enter key, another menu will appear containing a list of topics. Select any topic of your choosing and acquaint yourself with some of the many types of problems that can be solved with `MLAB`.

Otherwise, select the second option: Go to top-level MLAB, and click the CONTINUE button or strike the Enter/Return key.

You will then see an asterisk (`*`) that is your cue to type in instructions. The question is, "What instructions?" Start by using MLAB as a calculator. Type

```
3+5 <Enter>
```

and expect to get back `= 8` and a new asterisk. In all that follows, we shall just omit the `<Enter>`, but you should not, or nothing will happen. If your command is defective, MLAB will issue a sometimes cryptic diagnostic and put you into an edit mode to correct the command. Next try `((3+5+)+7)`. As with most computer languages, `*` is multiplication and `/` is division; the caret is used for exponentiation, with `Y = X ^3` giving the cube of `X` and `XRT = X ^0.5`, the square root.

MLAB can be used as a high powered calculator. The number of memory registers is potentially huge. Type `X1 = 8`, `X2 = 3`, and then `DX = X1 - X2`. Entering `TYPE DX` or `DX` will return the answer. Note: for clarity we will identify MLAB keywords by using all caps in this text. The software does not distinguish between upper and lower case.

The selection of basic built-in functions in MLAB is far more extensive than what is offered by any calculator. These include `SIGN(X)`, `INT(X)`, `CEILING(X)`, `FLOOR(X)`, `MOD(X,Y)`, `ABS(X)`, `SQRT(X)`, `EXP(X)`, `LN(X)`, `SIN(X)`, `COS(X)`. Define `X` and try some of these. Experiment with trigonometric identities and logarithms.

Note that all numerical variables in MLAB are 8-byte (64-bit) floating point. The full 10-byte, the short 4-byte, and integer formats are not directly available. This presents no practical problems. Rectangular arrays (vectors and matrices) are available, as will be illustrated in the demonstrations. The other data type implemented is alphanumeric (ASCII), which has an unusual facility with the `DO` statement.

User-defined functions are constructible, but these are limited to scalar (single number) outputs. These must be defined by a single line of MLAB code, except for the `... IF ... THEN ... ELSE ...` construction. By using `IF` and functions of functions, especially recursive functions, very elaborate functions can be defined. The necessary programming style is quite different from the usual compiler-based implementations. Examples will be given in the demonstrations.

Rational functions are good for constructing approximations, though not so good for doing the operations of calculus (differentiation and especially integration). Define a rational function by typing

```
FCT RATFINC(X) = (X^2+2*X-7)/(X^3-2*X^2+3*X-4)
```

You can type `RATFINC(0)` and `RATFINC(1)` and so on to build up a table that can be plotted. Do this now manually; later you will learn a way to compute and plot such a graph on your monitor.

Now let us consider the failure of the associative law of arithmetic. Define the functions

```
FCT SS1(A,B,C) = A+(B+C)
FCT SS2(A,B,C) = (A+B)+C
FCT DSS(A,B,C) = SS2(A,B,C) - SS1(A,B,C)
```

by typing in these statements as commands. In theory, the function `DSS()` is identically zero, but in computational fact it may not be. Try

```
X1 = -1.234567E12
X2 = 55.55555555555
X3 = 7.654321E11
DSS(X1,X2,X3)
```

The result will be `1.22070312E-4`, which is "small," but not zero. This is not a defect in MLAB or your computer, but in floating point implementations. Something similar will happen with compiler code. Exactly what happens can depend on the hardware, the operating system, and the software. You may get a different estimate of zero, including zero itself. A little experimentation, however, should enable you to find values that cause the associative law to fail.

There are some computations, however, that are very stable. Define the function

$$\mathrm{STEP1}(x) = \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{|x|}}}}}$$

in MLAB with

```
FCT STEP1(X) = SQRT(SQRT(SQRT(SQRT(SQRT(ABS(X))))))
```

This function has two fixed points x_f such that

$$x_f = \mathrm{STEP1}(x_f)$$

The value $x_f = 1$ is stable and $x_f = 0$ is unstable. This also is a compressed flow, since any interval is mapped by this function into a smaller interval.

Accelerate the compression with

```
FCT STEP2(X) = STEP1(STEP1(STEP1(STEP1(STEP1(X)))))
```

The acceleration can be increased by another level of compounding. Eventually you can get a function that maps the entire available floating point range of your computer into the value 1, except for $x = 0$. Try this.

Even a pocket calculator can compute this simple, discrete "time" dynamical system, by repeatedly pushing the square root key. This particular example is of no great interest in itself, but it illustrates in a simple way the concepts of stable and unstable fixed points, nonlinearity, and compressed flow. Models of considerable utility will entail internal computations much more complex than square roots, but MLAB will take care of that for you. And MLAB also can help with the analysis that can find these qualitative properties of your model that often are essential for design and calibration.

Now you can consider yourself at least a beginner in MLAB, as well as numerical analysis and nonlinear dynamics. And it wasn't so difficult, was it? Soon you'll be an expert, but it will require a little time, effort, and thought.

Chapter 2

PROGRAMMING IN MLAB

Programming is done in MLAB by creating ASCII files of MLAB statements. These files are never compiled into *.exe files as one would if using Fortran, Pascal, or C. They are executed from within MLAB by the command `DO filename`, where a file called `filename.DO` is available in the current directory.

This chapter will initiate the use of `*.DO` files as demonstrators. Read the text and then execute the `*.DO` file provided in the various sections of Appendix B.

Comments and pauses are included in the `*.DO` files, which will be written to the screen as the program is being executed. This enables the text to be more brief than was Chapter 1 To continue after a pause or the display of a graph just touch the Enter/Return key.

The `DO` statement also can be used with internally defined alphanumeric variables. The variable has to be a valid sequence of MLAB statements. This capability is used in many of the demonstrations. Where similar statements are need repeatedly, this form of `DO` can save a lot of effort.

2.1 More Numerical Analysis vs. Analysis

Polynomials are popular functions. For one thing the basic operations of calculus, differentiation and integration, are easy to do on polynomials.

The generalization of polynomials is the Taylor series, a sort of polynomial of infinite degree. If the argument is "small," the Taylor series is useful; when it is not, one must resort to more painful methods of analysis or numerical techniques.

Polynomials and Taylor series are best studied in the realm of complex numbers. There are always n roots of a polynomial of degree n, but some or all of these may be complex even if the

polynomial has only real coefficients. A function of a complex variable that is differentiable once is differentiable any number of times, which is not necessarily true for real-valued functions of real variables.

Differentiable (analytic) complex functions are conformal, they preserve angles in mapping one plane into another. (Complex numbers are 2-vectors, or a case of the set R^2, pairs of real numbers, with a peculiar multiplication defined.) Later we shall look at area preserving transformations that arise in Hamiltonian systems of differential equations. These and many other properties of models have limited or no connection with dynamics or physical reality. A more familiar example is continuity, which can be ascertained for mathematical functions, but never achieved in computational algorithms, and which atomic physics assures us never occurs.

For a more dramatic demonstration of the failure of the laws of arithmetic, let us construct `*.DO` files that evaluate a polynomial three different ways and then compare the results.

2.2 Legendre Polynomials

Legendre polynomials arise in the solution of Laplace's equation and related partial differential equations. They are used in constructing models of the gravitational field of the earth and other nearly spherical bodies, as well as other physical problems involving spherical boundary conditions.

One definition of the Legendre polynomial is

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n \qquad (2-1)$$

This is not very informative, but from it one can derive all the properties of these functions by much tedious algebra, which can be found in standard references.

There are (at least) two ways to evaluate any polynomial and three ways for a few special cases like the Legendre type. First there is "brute force." For any second-degree polynomial

$$p(x) = ax^2 + bx + c \qquad (2-2)$$

we can construct the MLAB functions

```
FCT P(X) = A*X^2 + B*X + C
```

The second alternative is the nested version

```
FCT PP(X) = C + X * (B + A*X)
```

For second-degree polynomials nesting usually makes no difference even for large absolute values of x, but for higher-degree polynomials it can succeed where the direct approach fails.

Recursion can be used for Legendre polynomials and several other kinds of functions. The specific formula is

$$P_n(x) = \frac{1}{n}[(2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)] \qquad (2-3)$$

but to get started we need $P_0(x) = 1$ and $P_1(x) = x$. In general recursions are highly accurate for a restricted range of arguments, but where they are good, they are very good.

The do-file `LEGENDRE.DO` B.11 computes and displays Legendre polynomials, using the MLAB built-in functions. `SLABLF.DO` B.22 uses the recursion formula in vector form to do the same thing. This is a more efficient way to proceed if you need the Legendre polynomials for lower orders too.

A third version is `KUGEL.DO` B.9, which computes and displays $P_n(\cos y), 0 \leq y \leq 90$ degrees. Trigonometric arguments actually appear in the application of Legendre functions to partial differential equations.

Rounding errors in evaluating polynomials are computed and displayed by `ROUNDOFF.DO` B.21 and by `POLLY.DO` B.18 using a high-order Legendre polynomial. Legendre polynomials will be used again to illustrate ideas from linear algebra and modern analysis.

Bessel functions arise in problems using cylindrical coordinates or circular boundary values. The first few orders are computed and displayed by `BESSEL.DO` B.1, using the MLAB function `BESSJ`.

Chapter 3

VECTORS AND MATRICES

One definition of vector is a quantity that has a direction associated with it, as well as a magnitude. For basic engineering and elementary mechanics that is intuitive and adequate. Two major generalizations are in current use, both being useful to scientists and modelers of all disciplines. Matrices are yet another generalization that has taken on a life of its own. MLAB has many standard operations and functions for matrices and vectors, as well as a lot of highly useful non-conventional ones. In what follows, scalar variables appear as italic lower case letters, e.g. a; vector variables appear as bold face lower case letters, e.g. $\mathbf{v}$; and matrices appear as bold face upper case letters, e.g. $\mathbf{A}$.

3.1 Vectors

To most users a vector is a column of real numbers, or occasionally complex numbers. To a computer a vector is an array of sequentially stored floating point numbers. For the pure mathematician a vector is a member of a "vector space," which includes operations, as well as some set of vectors.

Mathematics generalizes vector spaces to open-ended constructions with "infinite" dimension; this leads to a very rigorous approximation theory called functional analysis. By looking at some restricted abstract vector spaces, stronger theorems can be proven. In our demos we shall use an abstract vector space, orthogonal polynomials, to control numerical error for operations in a simple vector space of small dimension, where the vectors are columns of "real" (really floating point) numbers.

3.2 Matrices, Linear Algebra, and Modern Algebra

Matrices are rectangular arrays of real or complex numbers. One use they have is as realizations of linear transformations using matrix-vector multiplication

$$\mathbf{y} = \mathbf{A}\mathbf{x} \qquad (3-1)$$

The vector $\mathbf{y}$ has the dimension equal to the number of rows of matrix $\mathbf{A}$ and the dimension of vector $\mathbf{x}$ equals the number of columns. The study of these finite linear transformations is called linear algebra. For certain highly restricted classes of matrices, equation (3-1) can become a realization of abstract algebra. When $\mathbf{A}$ is restricted to rotation matrices, we have a group. Groups, rings, fields and the like comprise the subject matter of modern algebra. MLAB can be used to create computational realizations of these abstract systems. An interesting game to play is to figure out which few of these can be implemented without the disabilities created by rounding error.

Here is an example. Show that there are any indefinite (some would say "infinite") number of Pythagorean triangles. Like the 3-4-5 triangle, these are right triangles with all three sides rational. Show that they are dense, that is, any right triangle can be approximated arbitrarily well by a Pythagorean triangle. Finally, try to create a finite rotation group that is exact and closed under MLAB operations (64-bit floating-point arithmetic).

Most of linear algebra is simple and direct, but things can get ugly when we try to solve (3-1) for $\mathbf{x}$, given $\mathbf{y}$ and $\mathbf{A}$. When $\mathbf{y}$ has a dimension equal to $\mathbf{x}$, $\mathbf{A}$ is square and usually has an inverse such that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \qquad (3-2)$$

where $\mathbf{I}$ has 1 on its diagonal and 0 everywhere else. Then we can get

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} \qquad (3-3)$$

Computing the inverse matrix is not a simple task, but several good algorithms are available in MLAB.

Matrices provide linear mappings (transformations) from the vector space with columns of n numbers to that with m numbers. When $n = m$ and the matrix $\mathbf{A}$ has an inverse, the theory as presented in most texts on linear algebra is adequate. Non-square matrices or ones having no inverse ("singular") present some problems. In both cases the difficulty is caused by creating a mapping from a vector space of one dimension into a vector space of a different dimension.

A non-square matrix can be made square by filling in rows or columns with zeros. This creates a singular square matrix with the obvious property that certain dimensions are removed by collapsing their coordinates to zero. By changing the coordinate system, perhaps with just a rotation, the obviously singular matrix can become disguised. Numerical tests can help to identify the singularity, but rounding errors may make the results ambiguous. Sometimes a square matrix that theory asserts must be singular can be inverted quite nicely by a numerical algorithm.

Square matrices have eigenvalues and eigenvectors. These are found by solving the equation

$$a\mathbf{x} = \mathbf{A}\mathbf{x} \qquad (3-4)$$

What (3-4) means is that certain vectors are mapped into themselves with only a change of length (magnitude). (The eigenvalues a are commonly denoted by Greek lower case lambda, i.e. λ.) For singular matrices some or all the eigenvalues are zero.

Finding eigenvalues can be accomplished by solving

$$\det(\mathbf{A} - a\mathbf{I}) = 0 \qquad (3-5)$$

This is equivalent to finding the roots of a polynomial, which is not an easy chore for polynomials of degree larger than four. In fact, a good practical way to solve for the roots of a polynomial is to create a matrix that represents that polynomial by (3-5) and apply the MLAB algorithm for eigenvalues to get the roots.

Linear algebra can be done in a more general setting with the pseudo-inverse, where

$$\mathbf{A}\mathbf{A}^*\mathbf{A} = \mathbf{A} \qquad (3-6)$$

Sometimes this $\mathbf{A}^*$ is called the generalized inverse or Moore-Penrose inverse; a less well known codiscoverer is the geodesist Bjerhammer. Not only linear algebra, but much of the development of statistical adjustments, can be generalized by using the pseudo-inverse.

Presenting even a fraction of linear algebra is more than we can hope to do here, but some of these concepts are illustrated with the do-file `LALGEBRA.DO` B.10. Application is global for linear dynamical systems (Cf. `LINEAR.DO` B.13, `DIFFEQ.DO` B.4, `FILTER.DO` B.6, and `BOUNCE.DO` B.3) and local for nonlinear ones (Cf. `LIAPUNOV.DO` B.12).

3.3 Arrays

To computer users "matrices" may be just bins for storing numbers. The more fastidious pure mathematicians would prefer to have them called "arrays," in that case. Common in MLAB use is a long, narrow matrix with the independent variable ("time") in column 1 and state vectors ("time series") in the rest of the columns. This is how difference and differential equation solutions are tabulated; read, for example, about the MLAB functions `ITERATE()`, `INTEGRATE()`, `POINTS()`, `FIT()` and the operator `ON`. These functions will be illustrated later. `ON` was used in `LEGENDRE.DO` B.11 and `KUGEL.DO` B.9.

MLAB provides numerical implementations of the usual operations of vector analysis and linear algebra, as well as some mathematically unconventional column and row manipulations useful for data processing. When possible, use multiple indices [typically `(1:10)` for $1, 2, 3, ..., 10$] with the `ROW` or `COL` operators or the `ON` operator rather than the `FOR` loop to program array manipulation. This, together with recursive functions, defines a style for MLAB coding that makes it possible to implement almost any algorithm or procedure.

The do-file `LALGEBRA.DO` B.10 creates a square matrix of "random" numbers. Then its inverse is computed and tested. The matrix is made singular by setting the bottom row to all zeros. A pseudo-inverse is computed and tested. Eigenvalues for both matrices are computed and displayed in the complex plane, along with the imaginary axis and the unit circle. These geometric divisions of the complex plane have significant dynamical implications, to be explained in Chapter 4.

3.4 Functional Analysis

From basic geometry we know that vectors can be orthogonal, which means they lie at right angles to each other, or more generally, that the projection of one onto the other is a single point. A quick and easy test for this is having the "dot product" or inner product being zero

$$\mathbf{a} \cdot \mathbf{b} = 0 \tag{3-7}$$

This means the sum of the products of corresponding components is zero

$$a_1 b_1 + a_2 b_2 + a_3 b_3 = 0 \tag{3-8}$$

This easily extends to any number of dimensions.

The mathematical term "dimension" has many meanings, with the basic intuitive one being a sort of lowest common denominator. The three dimensions of classical physics are clear enough, but the

fourth dimension of relativity theory is a bit fuzzy. Time and space are not even unambiguously defined; they are nonholonomic.

An easy to understand example is elevation above sea level. The surfaces of equal potential energy are not exactly parallel or concentric, so elevation is not defined uniquely unless corrected with some formulas that take into consideration gravity variations. This has led some geodesists to develop their theories in tensor calculus.

In mechanics, classical or otherwise, the state space includes all dependent variables, both velocities and coordinates. (Sometimes this is called the phase space and state space is reserved for coordinates. But when orbital elements are used, they are functions of both position and velocity.) Most of the do-files have two-dimensional state spaces; but `LORENZ.DO` B.14 has a three-dimensional one; `VOLTERRA.DO` B.24 and `FITCYCLE.DO` B.7, four-dimensional; and `DIRKB.DO` B.5 and `VINTI.DO` B.23, six-dimensional.

Fractals are figures that may be two-dimensional, but have another kind of "dimension" that is not an integer value. Actually, they are sequences of figures. The fractals you may see in picture books are of a type that can be generated very quickly. Supercomputers are required to portray those that arise from the nonlinear equations of interest to most scientists. We return briefly to this subject in Chapter 9.

Functions with continuous domains can be used to create function spaces with infinite dimension. Engineers and physicists come into contact with Fourier series early in their careers and these display the basic properties.

The dot product is readily generalized to the definite integral of the product of two functions. By this definition sines and cosines over a full cycle are indeed orthogonal as well as being identical, except shifted by 90 degrees!

There are many families of functions that provide a basis for some interval or area or multidimensional region. The Legendre polynomials are orthogonal over $[-1, +1]$ (Cf. `LEGENDRE.DO` B.11, `KUGEL.DO` B.9, and `ORTHO2.DO`) B.17. A first priority of functional analysis is to create function spaces that are closed. The question is what properties a function must have to be "nice" (so that it can be represented by an infinite series of sines and cosines or Legendre polynomials or whatever). Here is an area where constructive and classic mathematics diverge on some technicalities, but rounding errors make the point mute for applications.

The do-file `ORTHO.DO` B.16 starts with an abominable nonorthogonal basis for the "nice" functions on the interval $[-1, +1]$, the powers $x^0 = 1$, x, x^2, x^3, etc., and creates the first few functions for an orthonormal basis by the Gram-Schmidt process. Powers are poor because they all look alike. Taylor series give accurate representations only over minute intervals. In the do-file the domain actually is a number of discrete points, which can be changed to get somewhat different results. These do not need to be evenly spaced.

With `ORTHO2.DO` B.17 one starts with the Legendre polynomials, orthogonal over the continuous

interval, and gets a similar set of functions orthogonal over the discrete subset of the domain. A simple test of the results shows them to be many times better than those from `ORTHO.DO` B.16. This is the result of starting closer to where we wanted to be, so the changes to be computed were small and created little rounding error.

Applying Gram-Schmidt, suitably generalized, to the powers on the continuous domain will yield the Legendre polynomials. This is a worthy exercise for those facile with analysis.

Another example of working from the continuous to the discrete is given in `DIFFEQ.DO` B.4, where an equivalent difference equation is obtained by continuous analytic continuation. Sometimes this is feasible for nonlinear systems too, but not always. The most important lesson from this is that models and computations and observable reality are three quite distinct things. Astute modelers work with both the discrete and the continuous, with both numerical simulations and analytic formalisms. "He who computes much, thinks little." With a little extra thought, you will be able to do most anything with MLAB on a PC.

Chapter 4

LINEAR DYNAMIC SYSTEMS AND MODELS

The term "linear" describes mathematical apparatus, not actual systems. If an equation has, say, two solutions and any linear combination of these is another solution, then the equations are linear. Note that these are general solutions; not enough information has been given to specify a unique solution.

Specific solutions are created by adding up a number of general solutions to get one that fits the starting values (initial conditions) for a dynamic system. Linear equations are wonderful because solutions to many kinds of problems can be constructed by superposition (add up many general solutions) and the mathematical theory is extensive.

Nonlinear equations, which comprise everything else, are highly individualistic and at best can be catalogued; there is no general approach. Solutions in terms of a few common functions or even infinite series may not exist or are computationally worthless.

Few things in the real world satisfy linearity absolutely or to a high degree of precision. This limits the usefulness of linear models and restricted possibilities for modeling before the computer era. However, there are a large number of "random walk" systems that have their nonlinearities masked by noise and damping. Some will be displayed in the next chapter, but first let us look at the simplest of linear dynamical models.

4.1 Difference Equations

Difference equations are based on functions of the integers, or sequences. What is great about them is that you do not need to plow through calculus to understand them. A little simple arithmetic

will turn the trick.

For dynamical systems it is imperative to have uniform spacing in time, but it is not necessary to use functions defined over all the integers; a small set will suffice. The simplest difference equation must be

$$x_{n+1} = ax_n \tag{4-1}$$

with a constant and n starting at some integer value, not necessarily 0.

To get things started we do need an initial condition, say

$$x_0 = 1 \tag{4-2}$$

and a value of a.

The most important properties of linear difference equations with constant coefficients can be derived from (4-1) almost effortlessly. The solution of (4-1) is

$$x_n = a^n x_0 \tag{4-3}$$

So if $a = 1$, $x_n = x_0$ for all n, which means no change. For $a = -1$ the value of x_n just oscillates between $-x_0$ and x_0. When $-1 < a < 1$ the value of x_n collapses to 0 as n increases. For values of a larger than 1 in magnitude, the solutions grow rapidly in magnitude, but may oscillate in sign.

It is easy to run examples on MLAB or a pocket calculator. How do you run (4-1) backwards to negative values of n?

These simple forms of behavior exhaust the possibilities of linear difference equations with constant coefficients, though some subtleties do appear for systems with 2 or more variables. The same is true of linear differential equations. Variable coefficients make life more interesting, but this is not the most useful or common generalization. The use of "inputs" or "right-hand sides" is; but that is saved for Chapter 5.

Generalizing (4-1) to more variables is done most easily using vector-matrix notation

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n \tag{4-4}$$

The vectors $\mathbf{x}_i$ are columns of numbers and the matrix $\mathbf{A}$ is a square array of matching size. Execute the product by taking the sum of the products of the numbers in rows of $\mathbf{A}$ with the

entries in $\mathbf{x}_n$. As an example consider the two-dimensional case $\mathbf{x}_i = (u_i, v_i)^T$ and a_{jk} is in row j and column k; then

$$u_{n+1} = a_{11}u_n + a_{12}v_n \qquad (4-5a)$$

$$v_{n+1} = a_{21}u_n + a_{22}v_n \qquad (4-5b)$$

Linear difference and differential equations with constant coefficients can be solved explicitly by matrix manipulations, with the theory being supplied by linear algebra. Some of this is discussed and illustrated in Chapter 3. MLAB users generally need not become real experts in the subject, but should be familiar with a few basic facts. Consult our reading program in Chapter 10.

There are functions of square matrices called eigenvalues that tell you all about the behavior of (4-4). These can be computed with the MLAB function `EIGEN()`. Look at the do-files `LALGEBRA.DO` B.10, `LIAPUNOV.DO` B.12, `FILTER.DO` B.6, `BOUNCE.DO` B.3, `DIFFEQ.DO` B.4, and `LINEAR.DO` B.13 for simple examples.

The difference equation in `DIFFEQ.DO` B.4 is designed to provide the same results as the differential equation in `LINEAR.DO` B.13, which in turn is almost identical to an example in the **MLAB 2-D Graphics Examples**. However, the coordinate axes have been rotated by one of the angles in a 3-4-5 triangle, so that the variables are abstract linear combinations rather than observables.

Such manipulations are common in celestial mechanics and some other disciplines that attempt analytic studies. They also can be useful for numerical work as done with MLAB, for example. A simple transformation sometimes can reduce numerical computations considerably and make a large or difficult job practical. Unfortunately, a lot of academic training and practical experience are often needed.

4.2 Differential Equations

Calculus has been made one of the *pontes asinorum* (bridges of asses), that is, barriers to entry, of the scientific and technical world. The subject is basically simple, but the textbooks and explanations are not. Its usefulness derives from the fact that theories can be constructed which describe the rates of change of processes with very simple functions, but the processes themselves cannot be specified so easily.

For a mathematical function f of time t, the derivative or rate of change is written as $\frac{df}{dt}$ or $\frac{df(t)}{dt}$ or sometimes $f\prime(t)$; another notation places a dot over the f, e.g. $\dot{f}$. The function is assumed to be defined over a useful interval of real numbers, so f may be used as a model for some process. Time

and the process are assumed to be meaningful to arbitrarily small divisions (unlimited resolution) for such continuous variable models.

No data stream has all the properties that can be ascribed to functions amenable to calculus; the use of such functions is to serve as an intermediate step to computational manipulations. Since the resolution of the computations is rarely identical to that of the process being modeled, using mathematical constructs with indefinitely great resolution is a convenient modeling technique. Numerical analysis builds the bridge between theory and computation; statistics and modeling technique relate theory to observations.

Almost anybody should understand that if $f(t)$ is a linear function of time

$$f(t) = at + b \tag{4-6}$$

then

$$\frac{df}{dt} = a \tag{4-7}$$

Computing derivatives of most functions is easy. Going from derivatives back to functions is more difficult. In cases where there are three or more interdependent functions, the difficulty escalates dramatically. Results usually have to be obtained by numerical manipulations, not calculus. In some cases even numerics can be confounded, as you will see in Chapter 9 on Chaos.

Many problems in physics require the rates of change of rates of change, or derivatives of derivatives. In mechanics these are accelerations, while derivatives are velocities. And the variables may be multiple and often are represented by vectors. For numerical work it usually is necessary to replace derivatives of derivatives by a system with two variables. If you have an equation involving $f\prime$ and $f\prime\prime = \frac{d(f\prime)}{dt}$, add a variable $g = f\prime$ and replace $f\prime\prime$ by $g\prime$.

Instead of (4-4) differential equations would look like

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}x \tag{4-8}$$

In all our examples, including Chapter 5, we will use matrices **A** of constants.

See if you can convert the damped linear system in the **MLAB 2-D Graphics Examples** to the one in `LINEAR.DO` B.13. Use the equations for a rotation in matrix or direct form. Note that the do-files run the equations back in time also, which is unstable.

4.3 Generalizations

In using MLAB you can make the entries in **A** functions of time; you still have a linear system. You can add a vector term $\mathbf{b}(t)$ to the right-hand side that depends explicitly on time; you're still linear. But when you make any part of **A** or **b** depend on **x**, you have gone nonlinear.

Linear difference and differential equations have long been popular in engineering. Laplace transforms are a favorite tool for those cases where there are "right-hand sides" or "inputs." When the input is "random noise" the Fourier transform and so-called times series methods are the natural tools. In most cases of the latter the linear operator, the ODE, need never be written down when only numerical work is being done.

What a model ODE can be used for is to do some of the typical time series analysis by formal manipulations, involving such horrors as contour integrals in the complex plane and Bessel functions of imaginary argument. This option offers alternatives to fast Fourier transforms when the data are not uniformly spaced or are small in number. The drawback is the need for the kind of analytic expertise shown in our reference JORDAN (1972).

The relations between linear ODEs and times series methods are developed in more depth in Chapter 5. The first parts of the do-files `FILTER.DO` B.6 and `BOUNCE.DO` B.3 show what happens when the input is null, so they are examples for this chapter; the final parts create simulations of filters and "random walk" times series by integrating ODEs. The simulated time series are then probed using Fourier methods.

MLAB will take nonlinearity in stride; in fact MLAB and most other differential equation software packages do not "know" whether they are working on a linear or nonlinear system. Any simplifications possible due to linearity (or symmetries) must be incorporated into the MLAB statements used. Actually, we recommend you do the opposite for complicated, nonlinear problems. Redundant calculations or symmetries are best used to monitor error growth; the use of integrals (a kind of symmetry) is illustrated in Chapter 8 on celestial mechanics. More on redundancies is in our reading program.

Modify the do-files with some simple experiments. Add nonlinear damping. Experiment with converting the result into a nonlinear difference equation, as done in `DIFFEQ.DO` B.4 for the linear case. The technique used there is a variation of continuous analytic continuation; the solution is found by Taylor series evaluated over steps that are short, but longer than those feasible with a numerical integration. This process does generalize to nonlinear systems and in those cases where higher derivatives are easy to compute, it can give high accuracy with economy.

Real cases of exponential growth (instability) are formidable forecasting problems. For one thing, the determination of the growth rate is sensitive to noise and various errors, but the big problem is that feedbacks cut in suddenly to control the process. The properties of these feedbacks usually are not known in advance and they may not be very consistent either. Nonlinear feedback is illustrated in Chapter 7.

The unit circle and imaginary axis are displayed on the graph of eigenvalues for the matrices. These are boundaries for stability regions for difference equations (4-4) and differential equations (4-8). Try to figure out which is which. [Hint: the answer for difference equations is easy to guess from (4-1, 2, 3).] For more details, begin the reading program in Chapter 10.

Chapter 5

LINEAR DYNAMIC SYSTEMS WITH INPUTS

The most useful linear dynamical models are those with "inputs" or "right-hand sides." Nonhomogeneous difference or differential equations is the mathematical term. In contrast, time varying coefficients are a less useful generalization; nonlinearity is more likely to be appropriate. Economics uses exogenous variables, that is, ones that do change, but in a way totally independent of the system being modeled. (Biology and celestial mechanics do too.) In cases where these are applied as variable coefficients it is easy to incorporate such structure in an MLAB model. Here we will restrict ourselves to the more common use as inputs.

5.1 Variation of Parameters and Filtering

Linear differential equations of the form

$$\frac{dx}{dt} = Ax + b(t) \qquad (5-1)$$

can be solved by variation of parameters. First solve the case for

$$b = 0 \qquad (5-2)$$

Then allow the adjustable constant parameters to be functions of time. These functions can be obtained by some tedious manipulations which usually are possible but rarely are practical. When the input b has a single, simple periodic component, just

$$b_i = a\sin(kt) + b\cos(kt) \qquad (5-3)$$

the output x will have this same frequency in some of the components, but with a shift in phase and change of amplitude. In those cases where the homogeneous equation decays to the origin, the steady state for the nonhomogeneous system will be just a sinusoid of this single frequency.

The equation (5-1) in that case will be a linear filter with input $b(t)$ and output $x(t)$. Placing another frequency in the input adds only that frequency to the output. Each frequency will have its own filtration characteristics, the amplitude change and phase shift.

Engineers traditionally have used Laplace transforms for such problems. These may do the trick even when the input is not linear combinations of sines and cosines. Where analytic techniques are feasible, it is a good idea to exploit them. It is also appropriate to do numerical solutions to check your work. MLAB can do the numerical solutions and also adjust the parameters in an analytic solution. Numerical simulations should always be run to test the modeling procedure.

The do-file `FILTER.DO` B.6 applies a single sinusoidal input to a stable differential equation with strong damping after exhibiting the solution with no input. Then random noise is used as an input. `BOUNCE.DO` B.3 does the same things with a linear system that oscillates as it decays.

5.2 Random Walks

Some of the most interesting inputs are random noise. The term "random" is generic, not precise. There are many statistical tests for predictability or lack of it, but no single index of randomness. Solutions of some fairly simple systems of ordinary differential equations can pass muster as "random," even though mathematics says that for given initial conditions, there is one and only one solution.

You may read oxymorons like "deterministic chaos" in the literature. This is a consequence of confusing computations with mathematics, and both of those with observational reality. A nasty differential equation can shuffle numbers just as effectively as any of the traditional schemes for generating "pseudo random" numbers. Such systems map simple initial configurations like a sphere or cube into sponge-like structures. The shuffle used in random number generators is just a difference equation version.

The most basic random walk is a running sum of a sequence of supposed random numbers. Of course there is no such thing as a single random number, only sequences of numbers that can be tested for various statistical properties. And there is no one test that rates "randomness" from 0 to 100 per cent.

Computer-generated sequences are often called "pseudo random" because they ultimately repeat and hence are completely predictable. But the period can be made long enough to provide an imitation of any natural process one might want to model. MLAB provides enough random sequence generators for most any simulation, all based on mapping a uniform ("boxcar") distribution into others, such as the Gaussian.

The do-file `RANDWALK.DO` B.20 first generates a sequence of normally distributed numbers. Rather than summing these, it produces a random walk by integrating over straight line segments connecting the numbers. The integrand might be called a linear spline. What is produced is a smoother curve than the original. Summation or integration is, in general, a smoothing procedure. This observation is the basis of all random walk models; some of the mathematics used to describe this can be arcane and almost bizarre, but implementation is easy with MLAB.

The do-files `FILTER.DO` B.6 and `BOUNCE.DO` B.3 use damped linear oscillators to convert the same kind of random inputs into smoother functions, that is, random walks. `FILTER.DO` B.6 exhibits only damping; the associated eigenvalues are both real and negative. In `BOUNCE.DO` B.3 the eigenvalues are complex conjugates, so there is a frequency associated with the process. Note that the former reduces the input sinusoid and the latter amplifies it. `BIZCYCLE.DO` B.2 has noise inputs for both first-order equations and displays its results as a phase plane diagram.

5.3 Time Series and Fourier Analysis

Random walks are usually not modeled with the sort of simulations (or better, emulations) displayed by `RANDWALK.DO` B.20, `FILTER.DO` B.6, and `BOUNCE.DO` B.3. Linear systems theory provides the broad and extensive framework. More narrowly focused books and articles talk about time series or linear filtering; geodesy uses the term collocation (from the theory of partial differential equations), because Laplace's equation is the basic linear system for gravity fields.

MLAB provides standard tools such as Fourier transforms and the cross correlation function `CROSSCORR()`. Other algorithms can be implemented easily using the standard and nonstandard matrix operations. Power spectra and autocorrelation functions for the random walks generated in `FILTER.DO` B.6 and `BOUNCE.DO` B.3 are computed and displayed.

Matching these (and other) statistical functions to those from data sets is the way to create models. There is no way to fit models to data and obtain a few parameters by multiple regression (least-squares adjustments). Interestingly enough, a linear filtering model will fit the observations exactly – and forecast for a short time thereafter.

Times series methods assume implicitly that there is no trend. To the Fourier transform a trend looks like a "sawtooth" wave. So trends should be removed before attempting Fourier analysis. Possibilities for trend models include polynomials (Cf. `POWER.DO` B.19 and `ORTHO.DO` B.16), the

Legendre polynomials (Cf. LEGENDRE.DO B.11, SLABLF.DO B.22, and KUGEL.DO B.9), and functions orthogonal on specified domains (Cf. ORTHO.DO B.16 and ORTHO2.DO B.17).

Sometimes it is a good idea to transform the data. For data that is always positive and is concave upward, the logarithm often does the trick. This generalizes to

$$y = \frac{(x^b - 1)}{b} \qquad (5-4)$$

(What happens at $b = 0$?)

When a trend is present in the data, it usually suggests a process that can be modeled by a more traditional "deterministic" method. From the trend model (or models) it may be possible to create a differential (or difference) equation model.

There has been a lot of attention paid to chaos now that "solutions" of nonlinear differential equations can be obtained numerically by computers and displayed on their color monitors (Cf. MITCH.DO B.15 and LORENZ.DO B.14). Identifying chaos in a data set is not so easy, especially when there is no highly developed theory to tell you it should be there. Even when chaos is present, a time series model may be the most effective.

Note that all our random walks have a stable equilibrium for the basic input-output system; when there is no input, the output dies out. The cause of chaos is one or more unstable equilibria. An unstable equilibrium scatters trajectories (solutions) all over the place. Collisions, as found in gas molecule dynamics, billiards, and roulette wheels, are extreme cases and wonderful destroyers of predictability.

A good indicator for chaos is transient periodicities. An interesting unstable equilibrium may have complex conjugate eigenvalues (Liapunov exponents – see LORENZ.DO B.14). If a sharp periodicity (a spike in the power spectrum) appears in one part of the data span, but disappears later, that is a good clue.

"Randomness" may be caused by collisions or deeply disguised unstable equilibria. In the case of celestial mechanics these unstable points can be located by perturbation methods. Other modeling tools, such as discrete Markov processes, may be the best choice. The matrix operations in MLAB are ideal for Markov processes.

Linear systems can be just as sensitive to initial conditions as nonlinear ones. Consider

$$x = x_0 \exp(1000t) \qquad (5-5)$$

Add the equation

$$y = y_0 \exp(-1000t) \qquad (5-6)$$

and you get a system that maps areas into equal areas of different shape. Start with a circle and you get an elliptical strand of ever increasing eccentricity. What an area-preserving nonlinear system will do is twist this strand into a blob like a ball of steel wool; sometimes the system maps into a distinct shape, not just said blob. These shapes are of considerable interest, since they indicate a kind of overall structural stability in spite of the unpredictability of specific state variables. (Cf. `LORENZ.DO` B.14).

The popular fractals, such as the Mandelbrot "set," are produced by mappings that rapidly separate stable and unstable trajectories that are near neighbors. Unfortunately, most systems of scientific interest require massive computation to produce fractal-like shapes.

In some cases the mappings decrease area (or volume, as in `LORENZ.DO` B.14, or hypervolume) to a figue of lower dimension, sometimes called the attractor. A better term, more consistent with traditional analysis, is sink. A point mass (in gravity systems) is an attractor, but definitely not a sink.

Random walks and chaotic systems are similar in that they have limited predictability. In other ways they are quite different. Both are intellectually stimulating because they represent peaks of divergence among mathematics, computation, and observable reality.

Chapter 6

A SIMPLE NONLINEAR MODEL: "HUBBERT'S PIMPLE"

To be useful a model need not be complex. When a simple model does not work at all, making it larger rarely makes it better. It is always best to start with a minimal model that captures the important qualitative features. This can be done quickly with MLAB. Even if it is necessary later to create a special purpose compiler code for regular forecasting and parameter improvement, MLAB can be used to develop the basic model.

Keep your MLAB development model. Additions to be made to a complex model often can be tested more readily with a basic core rather than the whole state-of-the-art development. Always run simulations before taking up the challenges offered by actual measurements.

6.1 The Mathematics of Resource Depletion

Exponential growth, most generally expressed by the differential equation

$$\frac{dx}{dt} = ax \qquad (6-1)$$

is not quite the simplest dynamical system. Uniform rotations or translations are. The difference equation

$$x_{n+1} = ax_n \qquad (6-2)$$

has the advantage that it requires no logarithms, let alone calculus, to solve and understand.

In the real world the behavior described by (6-1) or (6-2) can persist only for a limited time. In the case of growth, mathematical solutions grow too large for any actual process to keep pace. Numbers can grow ever smaller, but at various points exponential decay will drop below the limits of resolution, the size of subatomic particles, or anything else in the observable universe.

The impossibility of endless exponential growth is best illustrated by contemplation of the sequence of doublings: 1, 2, 4, 8, 16, ..., 2147483648, the last being the result of a mere 31 doublings. With an annual growth of only 1 per cent, a doubling takes about 70 years. So 31 doublings would require 2170 years.

World population is today about 5 billion, so if we assume a 1 per cent doubling, Adam and Eve must have lived only 2244 years ago, or in 253 B.C. That is not even the 4004 B.C. deduced by one scholar's reading of the scriptures. Perhaps we can deduce that the average annual population growth has been only 0.37 per cent, or about 1/9 the current value. All sorts of interesting games can be played with this compound interest arithmetic, but all it proves is that forecasting (or historical reconstruction) is a tricky business.

Physical limitations of some sort will contain exponential growth eventually. Estimating "eventually" is both challenging and important. Engineering jargon calls these limitations "feedbacks." In the simplest possible case, the feedback will depend only on the magnitude of the process. So we can try to generalize (6-1) to

$$\frac{dx}{dt} = ax - bx^2 \qquad (6-3)$$

The quadratic feedback is certainly simple enough, but that does not prove equation (6-3) is a useful model.

Equation (6-3) can be solved for $x(t, t_0, x_0)$ by a little bit of calculus. (Try it! Hint: the answer is coded in the `HUBBERT.DO` B.8 file.) One solution is the "S-curve" that is asymptotic to $x = 0$ for decreasing t and to $x = c$ for increasing t. This model serves rather well to reproduce the behavior of the depletion of a nonrenewable resource, such as metal ore, coal or oil. Metals often can be recycled, but fuels cannot, at least not as an energy source.

6.2 A Simulation

This humble special solution of (6-3) is called the continuous logistic. It has been known for several hundred years, but often is called "Hubbert's pimple." The "pimple" occurs when the time evolution of $\frac{dx}{dt}$, the rate of production, is plotted. It goes from zero, up to a peak, and down

to zero again. It is indeed a "bell-shaped" curve, but it decays more slowly than the well-known Gaussian distribution.

The geologist M. King Hubbert used this logistic to forecast the decline of U.S. domestic oil production in the mid 1970s. So did a lot of other people, but he was successful because of cautious calibration that observed that oil explorers were drilling more and enjoying it less. Oil per foot drilled was declining, giving an early warning of the inevitable.

Exploration for oil or other minerals is a complex process. In this simulation we assume frequency of discovery occurs in a Gaussian distribution. Size of discovery is set to a uniform distribution. Oil is pumped from the wells at an exponentially decaying rate. A small population of "wells" comprises the model. However, if we consider these entities to be oil fields, the number is not absurdly small.

The whole history of U.S. oil production is created, including the future. A logistic is fit to the data accumulated at different periods. Constraints that are highly simplified "geology" are applied. What can be seen is that the solutions are rather unstable before the curve hits its inflection point, but after that they home in quickly. Hubbert's contribution was to stabilize the forecast as soon as possible before reaching this critical juncture; that is not included in this simulation. What we see is a reproduction of the work of "the other guys": an inflated projection of future production that soon comes crashing down to reality.

Drilling offshore and in Alaska has not boosted net domestic oil production above Hubbert's projections. What we see here is the familiar effect of diminishing returns from ever more advanced (and costly) technology. Political, military, and economic machinations have caused crude oil prices to soar and sink. When prices sink, many low production "stripper" wells in North America close down forever, so the remaining oil is lost for future production.

Oil depletion has many factors that make it predictable. First of all, there is only so much, and no more. And there is only so much energy per gallon, no matter how improved refining technology becomes. Even the discovery of a really huge new field would not have postponed the day of reckoning very long; the exponential growth that characterizes the start of the logistic curve makes the lifetime of a nonrenewable resource relatively insensitive to the amount available. The discovery of several huge oil fields could have caused Hubbert's forecast to fail catastrophically, but this improbable event did not happen.

For other "limits-to-growth" problems there are too many feedbacks and all are too uncertain. There is plenty of natural gas and soft coal, so air and water may be more important constraints than fossil fuels. Toxic wastes, climate change, stress factors, technological change, and many other things effect economic and population growth. This is not to say that models should not be constructed, just that the value of making them too big and complex is nil.

Adding a term quadratic in x to (6-2) creates the discrete logistic. Its behavior over the range of "meaningful" parameters and starting values is as complex as the continuous case is simple. This

is demonstrated in Chapter 9 on Chaos. Chaos and "random walks" (Chapter 5) do not permit traditional modeling where one just fits parameters to observations and is done. There is need for analytic and numerical explorations, the study of critical points and statistical properties, and other indirect techniques. These too can be done with MLAB.

Chapter 7

A MORE COMPLEX NONLINEAR MODEL: VOLTERRA EQUATIONS

Nonlinear differential and difference equations display a wide variety of behaviors. Systems of two first-order differential equations are always in some sense solvable. Since the solutions are confined to a plane and cannot cross one another, it is possible to eliminate one of the variables and solve for the other by quadratures (simple integrations).

Sometimes this can be done with elementary functions, but in other cases special functions like elliptic integrals or elliptic functions (available in MLAB) are required. This is not always practical, however, since many other cases would require functions not yet studied and tabulated by mathematicians. Numerical solutions constructed with the help of analytic investigations are the best approach.

When you have three variables, or a single nonlinear difference equation, the game changes entirely. Trajectories (solutions) can cross over one another and become hopelessly knotted and tangled. They can spread out into a sponge-like structure. For some values of parameters and initial conditions there may be stable periodic solutions embedded in a sea of other solutions that develop into a sponge.

Both analytic and numerical work are essential for studying even the simplest nonlinear systems. In these final chapters we will illustrate the use of MLAB to facilitate both.

7.1 Simple Assumptions

In some cases biologists and ecologists have observed periodic fluctuations in natural populations of various species. The very simple system known as Volterra's equations provides a qualitative

model of this phenomenon.

This simple ecology has only two species, a number of prey denoted by x and predators, by y. The unchecked prey would grow at a rate a per unit time, but they are devoured by predators in proportion to the numbers of both. So the first equation is

$$\frac{dx}{dt} = ax - bxy \tag{7-1}$$

Predators increase in proportion to xy and die off at a fixed rate per unit time, c. So the next equation is

$$\frac{dy}{dt} = -cy + kxy \tag{7-2}$$

The simple model comprised of (7-1) and (7-2) has many sins of omission. For one thing it is continuous, but the number of individuals is always a positive integer. To be applicable the model requires large populations. The birth rate of the predators is bounded, but this is not accounted for in (7-2). Natural deaths among the prey, however, can be included by adjustment of a. For most species births are seasonal. And the food supply of the prey species surely is not limitless.

7.2 Mathematical Properties

For all its obvious flaws, the predator-prey model captures an important property. If you start with all the variables (x and y) and all the parameters (a, b, c, and k) positive, the variables stay positive and their change is cyclical. This can be shown by examining an integral of the system, which is a function which is constant for all points lying on a given solution, it may be written

$$f(x, y) = K \tag{7-3}$$

The curves (7-3) are closed loops in the first quadrant. They start out as ellipses (or circles) near the one equilibrium point, defined by

$$\frac{dx}{dt} = \frac{dy}{dt} = 0 \tag{7-4}$$

and develop into quarter circles (or ellipses) for cases getting close to the axes. We omit the analysis, which can be found in our readings, but illustrate the results in the do-file `VOLTERRA.DO` B.24.

7.3 Fitting to a Noise-driven System

The usual application of statistical adjustments assumes that the theory is perfect and that the observations are what is flawed. Most observations have significant errors and most theories have non-negligible omissions and unjustified implicit assumptions. The best approach is always to start simply, with a crude model, as we have here, and a naive adjustment of parameters and initial conditions.

Once you have captured the qualitative behavior, then you are ready to make the model bigger. Not before. Where there is a highly developed theory for the system under consideration, the first try can be correspondingly complicated. These are the exception rather than the rule. Early models for artificial satellite orbits started with as few as three parameters before progressing decades later to thousands.

Some mathematical theories assume the significant errors in models are small, but smooth, continuous effects. For a few problems in physics and astronomy, as for example artificial satellites, this is true. In most cases it is not. Noise inputs are the principal perturbations in many cases. As we saw with the business cycle emulator (Chapter 5), noise may be what keeps a system with damping "alive" and it can mask nonlinearities that we "know" must exist.

One might argue that the Volterra equations need more terms, because a small continuous perturbation could send the solutions spiraling outward or spiraling inward, as always happens in a homogeneous linear system with constant coefficients. Persistence of the cycle suggests to some a limit cycle, where solutions spiral towards a single periodic loop.

Simulations can be done with MLAB to determine what level of noise will mask a limit cycle mechanism, if one is present. Another thing to try is adding a third species, to see when the cycle breaks down and chaos occurs. We have mentioned the food of the prey. And there are diseases; a lot of modeling of epidemics has been done.

The complexity of models escalates rapidly with the addition of each new variable or parameter. It is not generally easy to decide what to add and what to omit. Chapter 9 and Chapter 10 provide some suggestions on model development.

The do-file `FITCYCLE.DO` B.7 adds noise terms to the Volterra equations and solves for the parameters using a noise-free model. Nonlinearities can produce biases in the resulting estimates of parameters.

Chapter 8

CELESTIAL MECHANICS AND HAMILTONIAN SYSTEMS

The motions of the planets have been studied for thousands of years by many cultures. Timekeeping, calendar construction, navigation, and astrology provided applications for the science and mathematics of the astronomers.

Modern science begins with Isaac Newton, who was able to describe the motions of the moon and planets with a small number of equations having a minimal set of parameters. Ancient theories of planetary motions contained large numbers of parameters and really were exercises in curve fitting, using three-dimensional trigonometric series.

Newton's laws of motion applied to falling apples and cannon balls, as well as celestial bodies. And by solving a very simplified set of equations for the sun and a single planet, one can obtain Kepler's laws. Before Newton's time, Kepler used the orbit of Mars to determine that planets moved in ellipses, with the sun at one focus, and that angular momentum was conserved.

By a rather circuitous path of development it was eventually discovered by Sir William R. Hamilton that Newton's equations could be expressed in terms of the partial derivatives of the energy function. Specifically,

$$\begin{aligned} \frac{dx_i}{dt} &= \frac{F}{y_i} \\ \frac{dy_i}{dt} &= -\frac{F}{x_i} \end{aligned} \qquad (8-1)$$

The functions $F(x_i, y_i), i = 1, ..., N$, is usually the energy or some generalization of it. In some modeling problems F may depend explicitly on time (e.g., the satellite of a rotating, asymmetric planet).

The curious symmetry of equations (8-1) has generated a vast literature and much speculation. For MLAB users these equations are attractive because the partial derivatives can be done symbolically by the `DIFF()` operator. As a specific example, consider the one-body problem, the sun and a planet of negligible mass. Then the position and velocity are

$$r = (x_1, x_2, x_3)^T \qquad (8-2)$$
$$\frac{dr}{dt} = (y_1, y_2, y_3)^T$$

and the energy (Hamiltonian function) is

$$F = \frac{1}{2}(\frac{dr}{dt})^2 - \frac{1}{r} \qquad (8-3)$$

with the first term being the kinetic energy and the second the potential energy per unit mass of the tiny planet. Using a one in the potential energy specifies what are called canonical units, which depend on the physical parameters of the objects involved and are not part of the SI or other metric or standard systems. However these units may eliminate many superfluous multiplications in a numerical integration. The form (8-1) often is called canonical too, but canonical (Hamilton's) equations may or may not use canonical units.

For our MLAB demonstration we integrate the system (8-1, 2, 3) numerically and display the variation of the integrals F and angular momentum

$$G = r\frac{dr}{dt} \qquad (8-4)$$

with time.

Mathematically, it is fairly easy to show that $\frac{dF}{dt} = 0$ and $\frac{dG}{dt} = 0$ (try it!). However, the inevitable errors created by a numerical solution of a system of differential equations produce variations of F and G along the computed solution. This is illustrated by the MLAB do-files `DIRKB.DO` B.5 and `VINTI.DO` B.23.

Check that `ERRFAC = 1.0E-5` before running `DIRKB.DO` B.5 or `VINTI.DO` B.23. This or larger values will cause the ellipse to fail to close in the graphics. `VINTI.DO` B.23 reduces `ERRFAC` by a factor of 100, which will cause the ellipse to close visually, but not numerically.

Some systems can be modeled by using more complex or different forms of F. Others, such as the atmospheric drag effects on an artificial satellite, cannot be included readily. Testing computations, especially things as cantilevered as numerical integrations, on special cases is essential.

Chapter 9

NONLINEAR SYSTEMS AND CHAOS

After being ignored for generations, the topic of nonlinear dynamics is now in vogue. Computers have made the subject accessible to those of less than genius level ability. Even the geniuses such as Henri Poincare and George D. Birkhoff did not make all that much progress on the subject. The contributions of topological dynamics are difficult for nonspecialists to understand.

Mathematics has succeeded in defining these modeling tools, that is, nonlinear difference and differential equations. What it cannot do is provide solutions. Simplifications and solutions of mathematical problems are possible only when there are some exact symmetries to exploit. Typical examples are energy and angular momentum, as in the Kepler problem. The integral for the simple Volterra equations is a generalization of energy.

Numerical methods, really usable only with a computer, can construct discrete samples of approximate solutions to nonlinear dynamical modeling equations. (But there is the method of continuous analytic continuation, illustrated and implemented with MLAB in `DIFFEQ.DO` B.4.) In some cases these can be used for modeling, parameter determination, and forecasting without any difficulty.

For some parameter values these models may exhibit chaos. Sometimes this is called "deterministic chaos," because a symptom is extreme sensitivity of future state values ("the solution") to initial conditions. (Bifurcation is extreme sensitivity to a parameter.) What should be kept in mind is that this is divergence between the ideal results of pure mathematics and the realities of floating point computations. The observable universe can be measured with quite limited precision, usually much less than the numerical precision available in MLAB (about 1 part in 10^{16}).

Modern physics has made limited resolution part of its canon, not just an economic or technical barrier to be hurdled by the next generation of instrumention. Where a chaotic model is deemed appropriate, there is a physical barrier to predictability. Not much can be accomplished by re-

sorting to special software with greatly extended precision, though this may be of some interest for experimental mathematics.

Even though "solutions," as commonly understood, are not available for chaotic systems, numerical simulations run on MLAB can be used to explore the territory, so to speak. There also are a lot of analytic techniques that can be used to determine the properties of models; using these is greatly facilitated with MLAB. A few examples will be illustrated in this chapter.

9.1 The Discrete Logistic

The subject of nonlinear dynamics was initially revived in the 1957-66 era to address the need for computing orbits of artificial satellites. Heroic attempts were made by Dirk Brouwer, Boris Garfinkel, Yoshiheda Kozai, John Vinti, Imre Iszak, and others to create a high-precision theory for satellite orbits. After Iszak died prematurely, this effort faded away and the "number crunchers" took over. Eventually the precision requirements of planetary exploration caused numerical integrations to displace analytic theories for the ephemerides of the major planets. A centuries-old enterprise all but ended.

Nonlinear dynamics reentered the mainstream of science in 1975, almost a decade after Izsak departed the scene, with the work of Mitchell Feigenbaum. Computers had predated artificial satellites by more than a decade, but they were of interest primarily to astronomy, geodesy, and a few other disciplines with traditions of massive computation.

Using computers meant dealing with programmers, key punchers, and a whole host of "computer professionals," or becoming one yourself. Most scientific users had to do both. But by 1975 terminals made computers directly accessible to users and central processors and memories were large and powerful enough to manipulate high-resolution graphics for video screens and plotters.

More physicists and a host of other scientists eagerly followed Feigenbaum into the *terra incognita* of nonlinear dynamics. The subject moved from obscure niches in pure mathematics and theoretical physics to a voguish centerpiece in the life sciences as well as the physical ones. Journals and books have proliferated like mushrooms.

Feigenbaum started with an incredibly simple model, the discrete logistic difference equation

$$x_{n+1} = ax_n(1 - x_n) \qquad (9-1)$$

The theory is exponential growth with a controlling feedback proportional to the square of the magnitude. This model has many wonderful properties, not the least of which is that it requires no calculus or advanced mathematics whatsoever. It is so simple that it can be run on a pocket

calculator (with the stroke of a single key on a programmable one). Analysis of much of its behavior can be achieved with no more than solving quadratic equations and other simple algebra.

For all this simplicity, it illustrates the basic properties of nonlinear systems. By contrast, the papers of the artificial satellite theorists are accessible only to a very few (and decreasing number of) specialists.

The do-file `MITCH.DO` B.15 propagates the solution of (9-1) and draws the now classic graph, with the horizontal axis being the value at step n and the vertical that for step $n+1$. Then x_{n+1} is reflected back to the horizontal axis by the line of slope 1. The quadratic nature of (9-1) creates a parabolic envelope for this trajectory.

Run some graphs using `MITCH.DO` B.15. Observe what happens as you raise a from 1 to 4. Watch the solutions evolve from collapse to periodic to doubly periodic to chaotic.

9.2 Lorenz's Equations

Lorenz's system predates the discrete logistic as a paradigm for chaos by a few years. However, it did not become widely known any sooner. The source of the equations is a model of a convection cell; partial differential equations are reduced to three ordinary differential equations by an approximation procedure.

The general form of the Lorenz equations is a restricted system of three ODEs, one linear, and the other two having a single quadratic term

$$\begin{aligned} \frac{dx}{dt} &= -ax + ay \\ \frac{dy}{dt} &= bx - y - xz \\ \frac{dz}{dt} &= -cz + xy \\ a > 0, b > 0, c > 0 & \end{aligned} \qquad (9-2)$$

The equilibrium points are easy to find (do it!), which is not true in general for nonlinear systems. Once they are found, the eigenvalues of the Jacobian matrix indicate whether the equilibrium is stable or unstable. The computation of eigenvalues isillustrated in a general setting in `LALGEBRA.DO` B.10 and for this particular dynamical model in `LIAPUNOV.DO` B.12.

Unlikely parameter values of $a = 10$, $b = 28$, and $c = 8/3$ generate the famous "butterfly" solution of (9-2). The numerical integration is started near the highly unstable equilibrium at the origin

in LORENZ.DO B.14. The two other equilibria are weakly unstable; the trajectory circles them like a moth does a flame, first one, and then the other, back and forth, again and again.

There are no stable equilibrium points to act as sinks, so the trajectory cannot dampen out. The trace of the Jacobian matrix is always negative, so Gauss's theorem asserts that this is a compressed flow. Trajectories cannot escape, so all approach some figure of zero volume, which, however, is not a single point! Sometimes this is called an attractor, but it really is a generalization of a sink.

Another kind of generalized sink is found in limit cycles. The best know is van der Pol's equation. Van der Pol and other investigators of modest nonlinear systems in the early part of the 20th century had to use tedious graphical methods and considerable ingenuity to get some results; now these can be obtained almost effortlessly with MLAB.

The Jacobian matrix is formed from all partial derivatives of the right-hand sides of (9-2) with respect to x, y, and z. Most texts on advanced calculus or vector analysis will tell you all you need to know about such things. In LIAPUNOV.DO B.12 the MLAB coding is suitable for getting the Jacobian matrix for any three-dimensional system; use is made of the MLAB capability for formal differentiation. This is easily generalized to more state dimensions, or to one or two.

Finding the equilibrium points of most nonlinear systems is a challenge. Use the MLAB function ROOT() and scan regions of interest for the roots of successive time derivatives. A mix of analytic work and numerical trial-and-error may be needed.

"Butterfly" comes from the observation that a butterfly flapping its wings in Beijing might trigger a thunderstorm in North America, because the weather system is so sensitive to initial conditions. This idea, of course, is silly. There is so much background noise in the real weather system and so much error in any conceivable model, that it is meaningless to assert that any insignificant event is the cause of something. But when you first run LORENZ.DO B.14, you will see that the figure looks like a butterfly.

More comments on nonlinear systems may be found in Section 5.2 and Section 5.3, where they are compared with random walks. The do-file FITCYCLE.DO B.7 generates a random walk using a nonlinear system with cyclic solutions. Numerical integrations of things like the classic Lorenz equations always have a significant random walk presence due to rounding errors.

Chapter 10

ADVANCING WITH MLAB

By reading the text, running the example `*.DO` files, and browsing through the **MLAB Reference Manual** you hopefully have become familiar enough with the MLAB syntax and the basics of modeling to strike out on your own. Start by modifying the do-files to create more complex simulations and models. A good practice is first to copy these example `*.DO` files to new names and rename the graphics windows before making other modifications.

Next you can use fragments of the `DO` file coding to create new models and simulations. Soon you will be ready to create models from scratch. There will always be some need to check the **MLAB Reference Manual**, except for the fortunate few with total recall, but even we absent-minded types will not have to look up every line we enter.

The reading program starts with a book suitable for those with almost no experience in modeling and then moves on to both the pragmatic and theoretical. The best idea is to start reading at a level that is easy, but not a waste of time. Implement what you learn in MLAB and move on. Build up your own library of `*.DO` files. Adapt a standardized scheme for coding types of variables so that it is easier to merge fragments into large programs. In some cases it may be necessary to use the "global replacement" command in your word processor to edit the fragments before merging.

10.1 Enhance the Demonstrations

Plot more of the special functions available in MLAB. Try to recreate and improve upon the graphics you find in reference books, such as JAHNKE and EMDE (1945).

Move the Legendre functions to a nonstandard domain of definition. Try some other types of orthogonal functions as first approximations to discrete orthogonal sets.

In the modeling simulations change the seed values of the random number generators. Try to find anomalous cases where the simulation departs from what is expected. If you succeed in doing that, try to make the simulation less responsive to unusual sequences of random numbers.

Try to orthogonalize some other functions on a discrete domain, as done in `ORTHO.DO` B.16 and `ORTHO2.DO` B.17. Sines and cosines are another system of continuous functions orthogonal on a certain interval; there are lots of other orthogonal functions to be found in reference books such as ABRAMOWITZ and STEGUN (1964). The associated Legendre functions (provided in MLAB) give rise to spherical harmonics, which are orthogonal over a sphere.

The linear system in `FILTER.DO` B.6 has already been generalized for you in `BOUNCE.DO` B.3, but there is no end to what you can add. More variables and different kinds of noise drivers are among the possibilities. In many cases the results could be obtained by formal analysis, but that would require many hours of effort and considerable specialized knowledge. With MLAB you can get easy-to-understand results in a few minutes.

When creating simulated data you may choose to add some observation errors, usually with a Gaussian distribution. Even when the errors are in the model and not the observations, doing a least-squares fit is the best way to start. Take `FITCYCLE.DO` B.7 and display the autocorrelation function of the residuals from the fit. (You may want to use 64 or 128 observations.) Rework the simulation so that there is Gaussian error in the observations, but no noise in the system. Look at the autocorrelation function (ACF) of the residuals in that case.

There are many models waiting in textbooks and journal articles for the chance to be coded and explored in MLAB. Those in the **MLAB Applications Manual** are already highly developed and ready to run.

10.2 A Reading and Practice Program

Those with little or no training in quantitative methods should start by scanning the book of EDWARDS and HAMSON (1989). Read carefully about any concepts with which you are not facile.

This **Introduction to MLAB** is based on **The Art of Modeling Dynamic Systems** (MORRISON, 1991). The examples chosen for the `*.DO` files span the Hierarchy of Dynamic Systems, that runs from static through deterministic and chaotic to stochastic. These models, as well as many others, and the basic mathematics needed for modeling are presented in this book in an intuitive, heuristic way. A few derivations are presented to illustrate the use of symmetry in analysis and to demonstrate symmetries that are not geometrically obvious.

One book can present only so much, so this one includes an annotated bibliography of 86 books and 24 papers that provide mathematics at various levels of rigor, more examples, and specialized

methodologies. The list of books is selective rather than exhaustive. There are more than 24 journals that publish articles potentially of interest to MLAB users. A list of over 1500 citations for MLAB is available at www.civlized.com. What is listed in the bibliography here are sources of models that can be implemented in MLAB.

Appendix A

BIBLIOGRAPHY

[1] Abramowitz, M. and I. A. Stegun, **Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables**, (National Bureau of Standards Applied Mathematics Series 55, U.S. Government Printing Office: Washington, DC, 1964). (Later printings include corrections. Also reprinted by Dover in paperback.)

[2] Aris, R., **Mathematical Modelling Techniques, Research Notes in Mathematics 24**, (Pitman Advanced Publishing Program: Boston, 1978).

[3] Beltrami, E., **Mathematics for Dynamic Modeling**, (Academic Press: Boston, 1987).

[4] Davis, H. T., **Introduction to Nonlinear Differential and Integral Equations**, (Dover: New York, 1962).

[5] Edwards, D. and M. Hamson, **Guide to Mathematical Modelling**, (CRC Press: Boca Raton, FL, 1989).

[6] Jahnke, E. and F. Emde, **Tables of Functions: with Formulae and Curves**, 4th ed., (Dover: New York, 1945). (Found in various other editions from German and American publishers.)

[7] Jordan, S. K., *Self-consistent statistical models for the gravity anomaly, vertical deflections, and undulations of the geoid*, **J. Geophys. Res.**, **77**, No. 20, 3660-3670, 1972.

[8] Luenberger, D. G., **Introduction to Dynamic Systems: Theory, Models and Applications**, (Wiley: New York, 1979).

[9] Mesterton-Gibbons, M., **A Concrete Approach to Mathematical Modelling**, (Addison-Wesley: Redwood City, CA, 1989).

[10] Morrison, F., **The Art of Modeling Dynamic Systems: Forecasting for Chaos, Randomness, and Determinism**, (Wiley-Interscience: New York, 1991).

[11] Morrison, F. and B. Douglas, *A comparison of gravity prediction methods on actual and simulated data*, **Geophysics**, **49**, No. 10, 1774-1780, Oct., 1984.

[12] Peschel, M. and W. Mende, **The Predator-Prey Model: Do We Live in a Volterra World?**, (Springer-Verlag: Vienna, 1986).

[13] Rietman, E., **Exploring the Geometry of Nature: Computer Modeling of Chaos, Fractals, Cellular Automata and Neural Networks**, (Windcrest Books: Blue Ridge Summit, PA, 1989).

[14] Thompson, J. M. T. and H. B. Stewart, **Nonlinear Dynamics and Chaos: Geometrical Methods for Engineers and Scientists**, (Wiley: Chichester, UK, 1986) (with corrections, 1987).

Appendix B

DEMONSTRATION *.DO FILES

Each section in this appendix contains a script of MLAB commands– referred to as an MLAB *do-file*, and the pictures generated by MLAB when executing the do-file. The MLAB commands in a do-file are printed in `UPPERCASE BOLD TYPEWRITER` font.

You can create and execute your own copy of any do-file in this Appendix as follows.

To begin, you should view this book with a program that can display .pdf files. Such programs include web-browsers that have appropriate *plug-in* programs–such as Mozilla Firefox, Macintosh Safari, or Microsoft Internet Explorer; or stand-alone .pdf file viewer programs–such as Adobe Acrobat, GSview, or Ghostscript.

When initially viewing this book with one of these programs, the program with which you view this book might not allow you to select and copy text with the mouse. In that case, it will be necessary to change some aspect of the .pdf file viewer program so that text selection and copying with the mouse is enabled. Check the entries in the *File*, *Edit*, *Options*, and *View* menus to enable text selection and copying with the mouse.

While you are viewing this book with the .pdf file viewer program, start the MLAB program as another process, and a plain-text editor program as yet another process.

On Windows computers, start MLAB by double-clicking on the MLAB icon on the desktop. After selecting `Go to top-level MLAB` from the startup menu and clicking the CONTINUE button, type the command:

```
EDIT FILE FILENAME
```

where `FILENAME` is the name of one of the do-files in this Appendix. For example, type:

```
EDIT FILE BESSEL.DO
```

to create a local copy of the first do-file.

MLAB will then launch the NOTEPAD editor program. A NOTEPAD window will appear on the screen, along with a warning dialog window with the message:

Cannot find the FILENAME file. Do you want to create a new file?

Click the YES button.

If you are viewing this **Introduction to MLAB** pdf-file in a web-browser window or Adobe Acrobat program, scroll down to the section of this Appendix containing the desired do-file and select the text in **UPPERCASE BOLD TYPEWRITER** font with the mouse, i.e. position the mouse at the start of the first line of do-file text, press the mouse button, and move the mouse to the end of the last line of do-file text. (The background of the selected text should appear in a different color.) With the text so highlighted, click on the *Edit* menu at the top of the web-browser or Adobe Acrobat window, and select *Copy*. This will copy the highlighted text to the Windows' clipboard buffer.

Once the clipboard buffer contains the selected text, activate the NOTEPAD editor window by positioning the mouse cursor on the NOTEPAD editor window's title bar and clicking the mouse button. Then click on the NOTEPAD editor window's *Edit* menu and select the *Paste* option. The contents of the Windows' cliboard buffer will be copied to the NOTEPAD editor window. Then click on the NOTEPAD editor window's *File* menu, and select the *Save* option. This will save the do-file to the MLAB working directory on the disk.

Finally, reactivate the MLAB command window by clicking on the MLAB command window's title bar, and type:

```
DO FILENAME
```

MLAB will then execute the named do-file.

On Macintosh OS7/8/and 9 systems, the same procedure as for Windows can be followed. Double-click the MLAB application icon and go to top-level MLAB. However, when you type the command:

```
EDIT FILE FILENAME
```

MLAB will launch the TEACHTEXT on Macintosh OS7 systems, or the SIMPLETEXT editor program on Macintosh OS8 and 9 systems, instead of the NOTEPAD editor program. When

the TEACHTEXT or SIMPLETEXT program window appears, the same copy, paste, and save operations described previously for Windows, can be used to create a local copy of the desired do-file.

On Linux with X-Windows and Macintosh OSX systems, MLAB does not support the `EDIT FILE` command. However once you start MLAB, you can manually start a text editor. After you dismiss the MLAB startup menu and get the MLAB prompt, strike Ctrl-Z (i.e. press the Ctrl key and simultaneously strike the Z key). This will suspend the MLAB process and leave you with an XTERM bash shell prompt (on Linux) or a Darwin bash shell prompt (on Macintosh OSX).

You can then launch a text editor, such as vi or emacs, by issuing the command:

```
emacs FILENAME
```

or

```
vi FILENAME
```

Use the same copy, paste, and save procedure described for Windows to create a local copy of the desired do-file. Once you have created and saved the do-file, terminate the editor program. When the bash shell prompt appears, strike *fg* to resume the suspended MLAB session.

If your local copy of a do-file fails to execute without error, check that you have copied all of the MLAB commands to the do-file; partial lines or incomplete do-files will generate MLAB errors when executed. Also note, that on Linux with X-Windows and Macintosh OSX systems, filenames are case-sensitive. Be certain that the name of the do-file you created on disk is the same name used in the MLAB `DO` command. You can delimit the filename in an MLAB `DO` command with double quotes, i.e.

```
DO "FILENAME"
```

to remove all doubt as to which do-file is to be executed.

Each of the do-files in this Appendix begins with the same two MLAB commands. The first command: `ECHODO = 3`, causes subsequent commands in the do-file to be printed in the MLAB command window and in the MLAB log file.

The second command: `RESET`, causes all previous user-defined data objects, including scalar variables, matrices, functions, constraints, windows, etc., to be deleted. However, note that various MLAB control variables, such as `ERRFAC` which controls the precision of the ordinary differential equation solver, are unaffected by the `RESET` command. If an MLAB control variable is to be restored to its default value, an explicit assignment statement must be given. For example, to restore `ERRFAC` to its default value, type:

```
ERRFAC = 0.001
```

Comments in a do-file, i.e. lines with descriptive text to help the reader understand the intent of subsequent commands, are ignored by the MLAB interpreter. Comments are delimited by `/*` and `*/`, following the convention of the C-language.

The `DRAW` and `TITLE` statements cause curves and titles to be drawn in a separate MLAB graphics window. An MLAB graphics window does not appear on the computer screen until the `VIEW` command is given. An `UNVIEW` command will remove the MLAB graphics window from the screen. A `BLANK` command causes a previously visible curve, title, or window to be invisible; an `UNBLANK` command causes a blanked curve, title, or window, to reappear.

Consult the **MLAB Reference Manual**, **MLAB Applications Manual**, and **MLAB Graphics Examples** for more complete explanations of these and other MLAB commands.

All figures generated by a do-file are deferred to the end of the section containing the do-file.

B.1 BESSEL.DO

The do-file `BESSEL.DO` creates a graph of the Bessel functions of order 0, 1, 2, 3, 4, 5, and 6.

```
/* FILE: BESSEL.DO */
ECHODO = 3
RESET

/* THIS BESSEL.DO FILE CREATES A GRAPH OF BESSEL FUNCTIONS
 ORDERS 0-6 */
/* SELECT A TITLE IN GERMAN GOTHIC: */
TOP TITLE "Zylinderfunctionen" FONT 22, AT (0., 0.95)

/* CREATE THE ARGUMENTS LIST FOR THE BESSEL FUNCTIONS */
NSTEPS = 301
NULLZEHN COL 1 ROW (1:NSTEPS) = (0:10!NSTEPS)
BESSELFC COL 1 = NULLZEHN COL 1

/* GENERATE BESSEL FUNCTIONS DEGREES 0-6 IN MATRIX
 BESSELFC.*/
FCT BESSJF(X,Y) = BESSJ(X,Y)
FOR I = 0:6 DO \{
 NULLZEHN COL 2 = I;
 BESSELFC COL(I+2) = BESSJF ON NULLZEHN}
```

```
/* GRAPH THE BESSEL FUNCTIONS IN DISTINCT COLORS: */
DRAW BESSELFC COL(1,2) COLOR GREEN
DRAW BESSELFC COL(1,3) COLOR BROWN
DRAW BESSELFC COL(1,4) COLOR RED
DRAW BESSELFC COL(1,5) COLOR ORANGE
DRAW BESSELFC COL(1,6) COLOR VIOLET
DRAW BESSELFC COL(1,7) COLOR AQUA
DRAW BESSELFC COL(1,8) COLOR ROSE
BESSW = W
VIEW BESSW
```

Window `BESSW` is shown in Figure B.1.

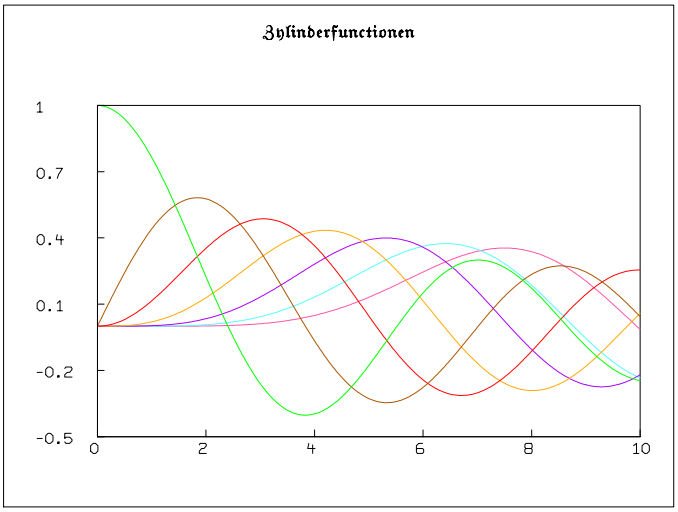

Figure B.1: Window `BESSW`.

B.2 BIZCYCLE.DO

The do-file BIZCYCLE.DO creates a vector of uniform random numbers from the interval [-1,+1]. Then a business cycle model is created by using a linear interpolation of these numbers as input to a damped harmonic oscillator.

```
/* FILE: BIZCYCLE.DO */
ECHODO = 3
RESET

/* THIS FILE CREATES A VECTOR OF UNIFORM RANDOM
 NUMBERS. A BUSINESS CYCLE MODEL IS CREATED BY
 USING A LINEAR INTERPOLATION OF THESE NUMBERS
 AS INPUT TO A DAMPED HARMONIC OSCILLATOR.*/
JAHRE=50; NSTEPS = 1 + 12*JAHRE
M COL 1 = 0:JAHRE!NSTEPS
START = RAN(1009,-1,1)
FCT BOXCAR(WX) = RAN(0,-1,1)
M COL 2 = BOXCAR ON M COL 1
MM COL 1 = M COL 1
MM COL 2 = BOXCAR ON MM COL 1

TOP TITLE "Uniform Random Sequence" FONT 8
IMAGE .18 TO .9, .13 TO .8 FFRACT COLOR BLACK
DRAW M COL(1,2) COLOR BROWN
GRS = W
VIEW GRS
BLANK GRS

FCT Y1'T(T) = A11*Y1 + A12*Y2 + B29*LOOKUP(MM,T)
FCT Y2'T(T) = A21*Y1 + A22*Y2 + B29*LOOKUP(M,T)
A11 = -9/250; A12 = 119/125
A21 = -131/125; A22 = -8/125
B29 = 6/5
INITIAL Y1(0) = 0.8
INITIAL Y2(0) = -0.6
SPIN = "N=INTEGRATE(Y1,Y2,0:JAHRE!NSTEPS)"
DO SPIN;
N1 = FLOOR(NSTEPS/3); N2 = FLOOR(2*NSTEPS/3)
DRAW N COL(2,4) ROW(1:N1) COLOR BROWN
DRAW N COL(2,4) ROW(N1:N2) COLOR GREEN
DRAW N COL(2,4) ROW(N2:NSTEPS) COLOR BLUE
```

```
DRAW (0&'0) LT NONE PT CROSSPT COLOR ORANGE
DRAW (0&'0) LT NONE PT CIRCLE COLOR ORANGE
IMAGE .23 TO 0.9, .13 TO .8 FFRACT COLOR BLACK
TOP TITLE "Business Cycle Emulation" FONT 34
BCYCLE = W
VIEW BCYCLE
BLANK BCYCLE

FCT RADD(T1,YY1,VV1,YY2,VV2)=SQRT(YY1*YY1+YY2*YY2)
TUPI=PI+PI
FCT ATAN4(YY,XX)=IF YY>0 THEN ATAN2(YY,XX) \
 ELSE TUPI+ATAN2(YY,XX)
FCT ATAN4D(YY,XX)=MOD[180*ATAN4(YY,XX)/PI,360]
FCT PHASE(T1,YY1,VV1,YY2,VV2) = ATAN4D(YY2,YY1)
RR = RADD ON N
FIZZ = PHASE ON N
N COL 6 = RR
DEL RR
N COL 7 = FIZZ
DEL FIZZ
TOP TITLE "Radial Coordinate" FONT 9
DRAW N COL(1,6) COLOR BROWN
BIZRAD = W
VIEW BIZRAD
BLANK BIZRAD

TOP TITLE "Phase Angle" FONT 9
DRAW N COL(1,7) COLOR BROWN
BIZFAZ = W
VIEW BIZFAZ
```

Window `GRS` is shown in Figure B.2.
Window `BCYCLE` is shown in Figure B.3.
Window `BIZRAD` is shown in Figure B.4.
Window `BIZFAZ` is shown in Figure B.5.

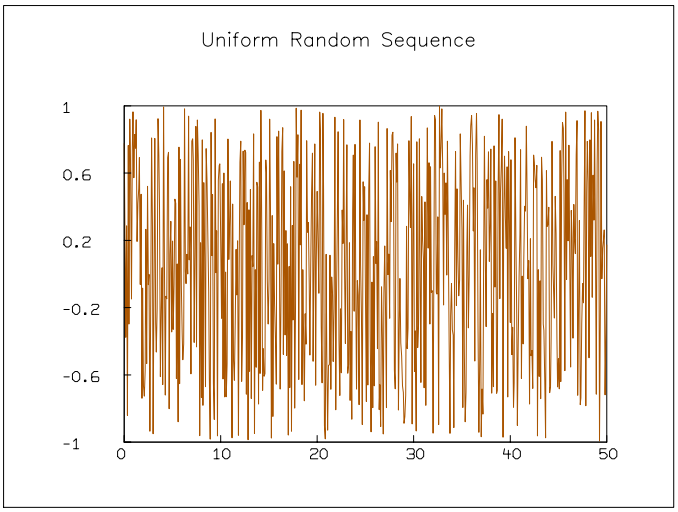

Figure B.2: Window `GRS`.

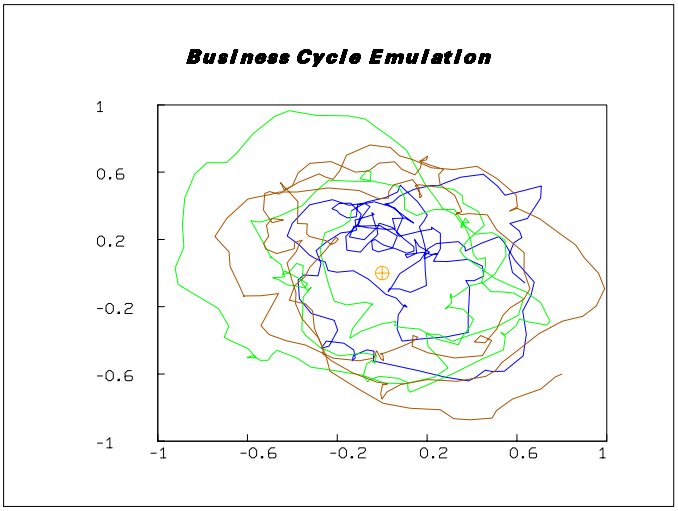

Figure B.3: Window `BCYCLE`.

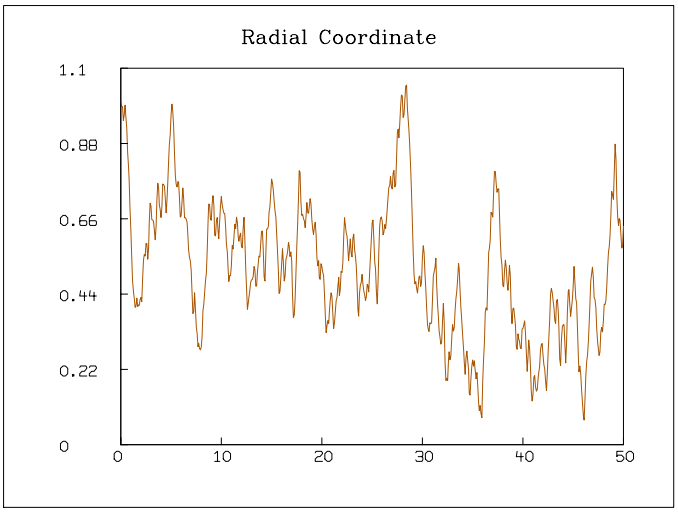

Figure B.4: Window `BIZRAD`.

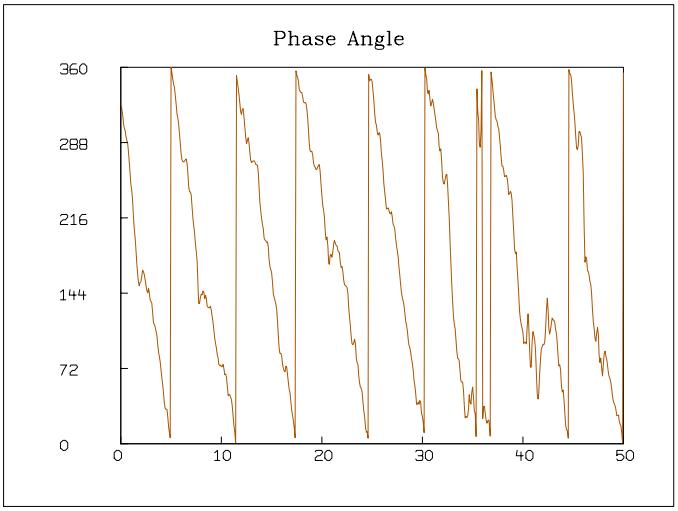

Figure B.5: Window `BIZFAZ`.

B.3 BOUNCE.DO

The do-file BOUNCE.DO defines and solves a linear ode system with 3 distinct inputs. After the initial impulse dampens out, the solution with a cosine input becomes a sinusoidal output of different magnitude and shifted in phase. Linear odes act like filters, so they can provide interpretations of noisy and complex systems.

The model here is a damped linear system (ode) having complex conjugate eigenvalues, with inputs F=0, cosine, and noise. In economics this is the cobweb model. cf. BIZCYCLE.DO B.2.

```
/* FILE: BOUNCE.DO */
ECHODO = 3
RESET

/* THIS IS BOUNCE.DO IT IS A STABLE, LINEAR ODE WITH
 3 DISTINCT INPUTS. AFTER THE INITIAL IMPULSE DAMPENS
 OUT, THE SOLUTION WITH A COSINE INPUT BECOMES A
 SINUSOIDAL OUTPUT OF DIFFERENT MAGNITUDE AND SHIFTED
 IN PHASE. LINEAR ODES ACT LIKE FILTERS, SO THEY CAN
 PROVIDE INTERPRETATIONS OF NOISY AND COMPLEX SYSTEMS.

 THE MODEL HERE IS A DAMPED LINEAR SYSTEM (ODE) HAVING
 COMPLEX CONJUGATE EIGENVALUES, WITH INPUTS F = 0, COS,
 AND NOISE. IN ECONOMICS THIS IS THE COBWEB MODEL.
 CF. BIZCYCLE.DO. */

/* FORMULATE 2ND-ORDER ODE AS TWO 1ST-ORDER ONES: */
FCT Y'T(T) = A11*Y - U + F(T)
FCT U'T(T) = Y

/* WE MUST HAVE A11 < 4.  */
A11 = -0.3
FCT F1(T) = 0.72*COS(A*T); A = 1.27
INITIAL U(0) = 0.8
INITIAL Y(0) = -0.6
SPIN = "N=INTEGRATE(U,Y,0:50!NSTEPS)"

/* CHOOSE NSTEPS = 2^N, N AN INTEGER, FOR FFTS.*/
NSTEPS = 2^9
FCT F(T) = 0
DO SPIN
TOP TITLE "Classic Cobweb" FONT 7
```

```
DRAW N COL(1,2) COLOR BROWN LT DASHED
DRAW N COL(1,4) COLOR RED LT LDASH
COBWEB=W
VIEW COBWEB

BLANK COBWEB
FCT F(T) = F1(T)
DO SPIN
N COL 6 = F ON (N COL 1)
DRAW N COL(1,2) COLOR BROWN
DRAW N COL(1,6) COLOR GREEN LT DASHED
TOP TITLE "Cosine-Driven Cobweb" FONT 7
WEBCYCLE = W
VIEW WEBCYCLE

BLANK WEBCYCLE
START=NORMRAN(EXP(1))
FCT F2(ARG)=NORMRAN(0)
M COL 1 = N COL 1
FCT F(T) = LOOKUP(M,T)
M COL 2 = F2 ON (M COL 1)
DO SPIN
TOP TITLE "Noise-Driven Cobweb" FONT 7
DRAW N COL (1,2) COLOR BROWN
MARKET = W
VIEW MARKET

BLANK MARKET
AAA ROW 1 COL 1 = A11
AAA ROW 1 COL 2 = -1
AAA ROW 2 COL 1 = 1
AAA ROW 2 COL 2 = 0
EA = EIGEN(AAA)

/* HERE ARE THE EIGENVALUES FOR THE FILTER: */
TYPE EA COL 1 ROW(1,2)
TYPE EA COL 2 ROW(1,2)
PAUSE
DEL AAA

/* SELECT FINAL STATES; GET FFT: */
TS COL 1 = N COL 1
```

```
TS COL 2 = N COL 2
TS COL 3 = 0; DEL N
FFT = DFT(TS)
DEL TS

/* GET POWER SPECTRUM = FFT*CONJ(FFT) */
FFT COL 4 = (FFT COL 2)*'(FFT COL 2) + \
 (FFT COL 3)*'(FFT COL 3)
DRAW FFT COL(1,4) COLOR GREEN
TOP TITLE "Power Spectrum Market Simulation" FONT 7
PSMARKT = W
VIEW PSMARKT

BLANK PSMARKT
FFT COL 5 = LN ON (FFT COL 4)
DRAW FFT COL(1,5) COLOR GREEN
TOP TITLE "LN of Market Power Spectrum " FONT 7
LNPSMARK = W
VIEW LNPSMARK

BLANK LNPSMARK
/* GET ACF FROM INVERSE FFT ON SPECTRUM: */
FFT COL 2 = FFT COL 4
DEL FFT COL(4:5)
FFT COL 3 = 0
ACF=IDFT(FFT)
MAXACF=MAXV(ACF COL 2)
ACF COL 2 = (ACF COL 2)/MAXACF
LONG=FLOOR(NROWS(ACF)/2)
DRAW ACF COL (1,2) ROW(1:LONG) COLOR GREEN
AXE COL (1:2) ROW (1:2) = 0
AXE COL 1 ROW 2 = 25
DRAW AXE COLOR RED
TOP TITLE "ACF for Market Simulation" FONT 7
ACFGRAPH=W
VIEW ACFGRAPH
```

Window COBWEB is shown in Figure B.6.
Window WEBCYCLE is shown in Figure B.7.
Window MARKET is shown in Figure B.8.
Window PSMARKT is shown in Figure B.9.
Window LNPSMARK is shown in Figure B.10.

Window `ACFGRAPH` is shown in Figure B.11.

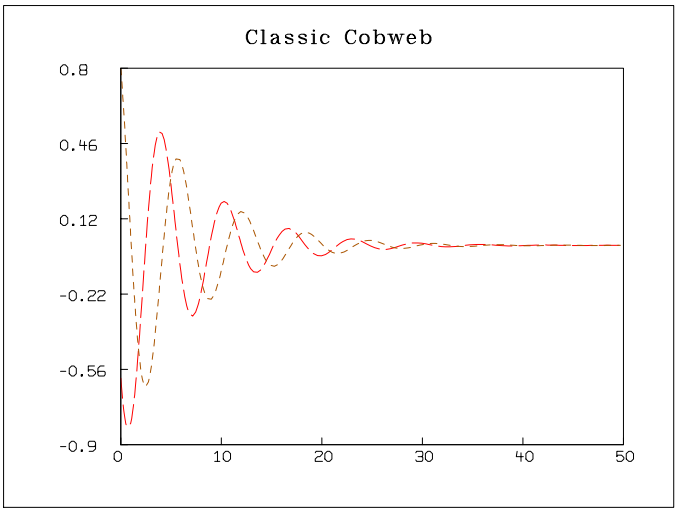

Figure B.6: Window `COBWEB`.

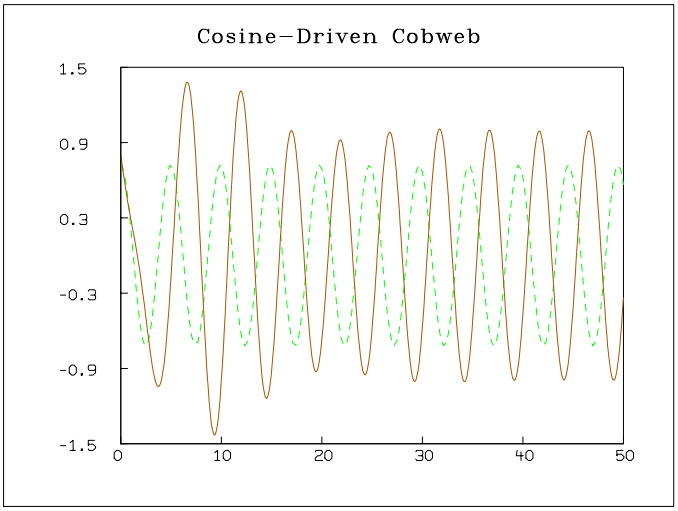

Figure B.7: Window `WEBCYCLE`.

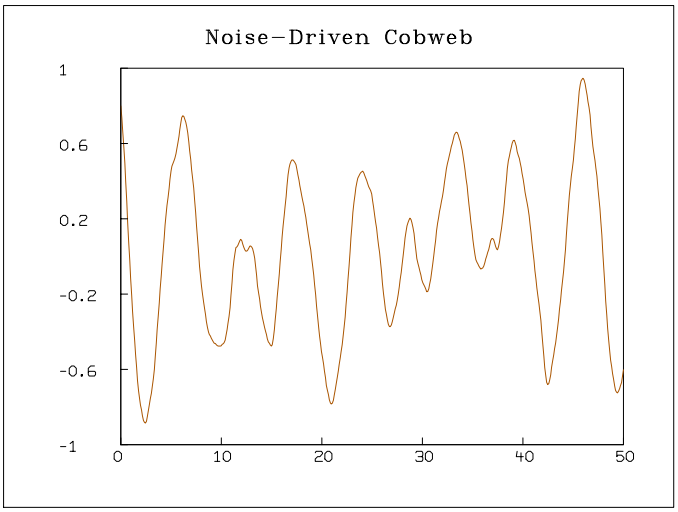

Figure B.8: Window `MARKET`.

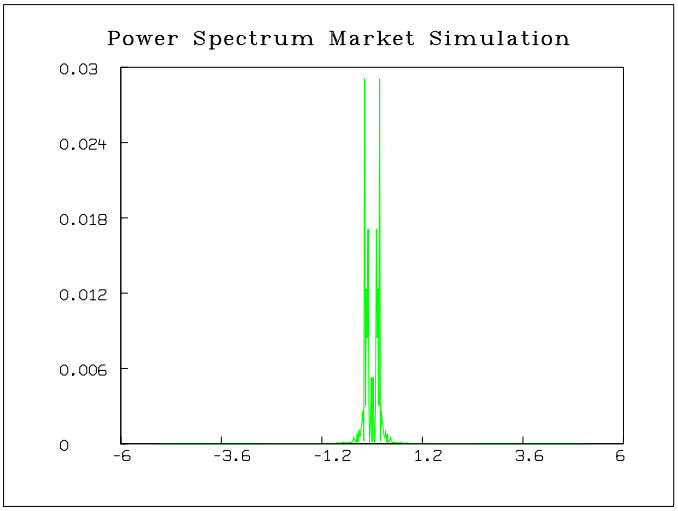

Figure B.9: Window `PSMARKT`.

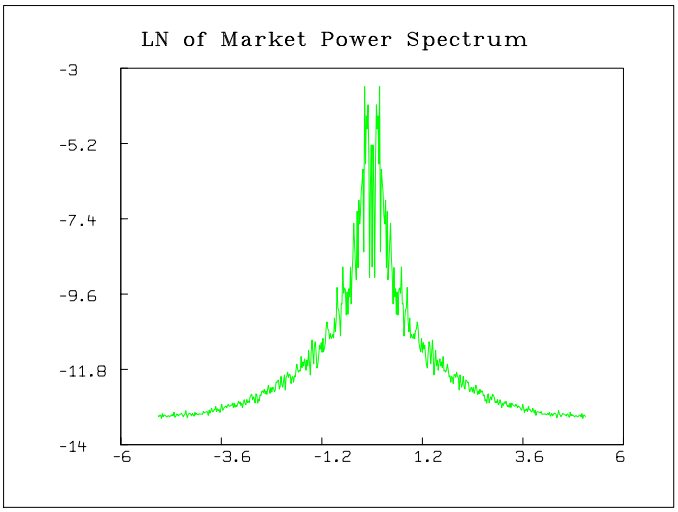

Figure B.10: Window `LNPSMARK`.

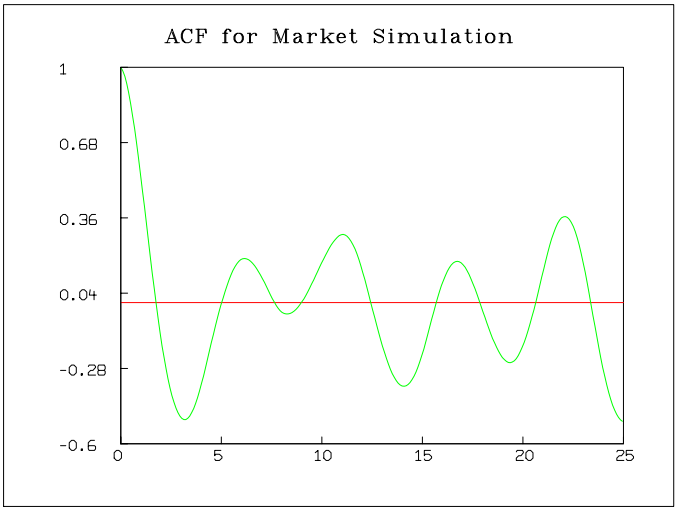

Figure B.11: Window `ACFGRAPH`.

B.4 DIFFEQ.DO

The do-file DIFFEQ.DO defines and solves a linear difference equation. The solution lurches out and lurches in according to a system matrix. The same equation is run backward by inverting the system matrix. Start with the system matrix for an ode as in LINEAR.DO B.13 and get the system matrix for an identical difference equation by using a Taylor series. This is a form of continuous analytic continuation.

```
/* FILE: DIFFEQ.DO */
ECHODO = 3
RESET

/* THIS IS DIFFEQ.DO. IT LURCHES OUT AND LURCHES IN
 ACCORDING TO A LINEAR DIFFERENCE EQUATION WITH
 CONSTANT COEFFICIENTS. THE SAME EQUATION IS RUN
 BACKWARD BY INVERTING THE SYSTEM MATRIX. START
 WITH THE SYSTEM MATRIX FOR AN ODE AS IN LINEAR.DO
 AND GET THE SYSTEM MATRIX FOR AN IDENTICAL
 DIFFERENCE EQUATION BY USING A TAYLOR SERIES.
 THIS IS A FORM OF CONTINUOUS ANALYTIC
 CONTINUATION.*/

FCT X1'T(T) = A(1,1)*X1 + A(1,2)*X2
FCT X2'T(T) = A(2,1)*X1 + A(2,2)*X2
A(1,1) = 27/250; A(1,2) = -107/125
A(2,1) = 143/125; A(2,2) = 24/125
ASTAR = SHAPE(2,2,LIST(1,0,0,1))
IDENT = ASTAR
B = ASTAR; EYE = 1;
TYPE B;
PAUSE

DELT = 1.; TOL = 1.E-11
S1= "B=(DELT/EYE)*A*B; ASTAR = ASTAR+B"
S2 = "TEST = MNORM(B,0)/MNORM(ASTAR,0)"
FOR I = 1:25 DO { \
 DO S1; DO S2;
 IF TEST<TOL THEN BREAK \
 ELSE EYE=EYE+1 }
TYPE EYE, TEST, ASTAR;
PAUSE
```

```
A = ASTAR
INITIAL X1(0) = 0.8
INITIAL X2(0) = -0.6
STIR = "N=ITERATE(X1 'T,X2 'T,51)"
DO STIR
ORIGIN="DRAW (0& '0) LT NONE PT CROSSPT COLOR RED"
DO ORIGIN
DRAW N COL(2,3) COLOR BROWN
TOP TITLE "Lurch Outward" FONT 34
OUTBOUND = W
VIEW OUTBOUND

BLANK OUTBOUND
/* EIGENVALUES OF THE SYSTEM MATRIX: */
LBL="Eigenvalues of the system matrix:"
EI="EA=EIGEN(A); TYPE LBL; TYPE EA COL (1:2) ROW(1,2)"
EIER="TYPE LNL; FOR I=1:2 DO { TYPE MNORM(EA ROW (1:2) COL I) }"
LNL="ABSOLUTE VALUES ARE:"
DO EI; DO EIER;
A = INV(ASTAR);
DO STIR
DO ORIGIN
DRAW N COL(2,3) COLOR GREEN
TOP TITLE "Lurch Inward" FONT 34
INBOUND = W
VIEW INBOUND

DO EI
DO EIER
PAUSE
```

Window `OUTBOUND` is shown in Figure B.12.
Window `INBOUND` is shown in Figure B.13.

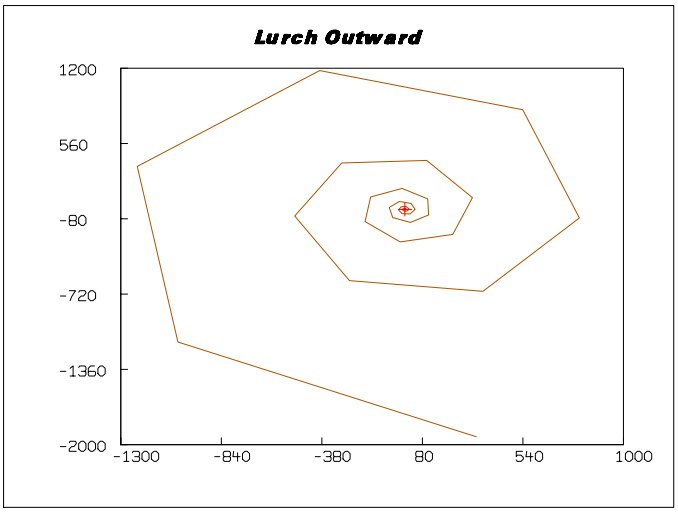

Figure B.12: Window `OUTBOUND`.

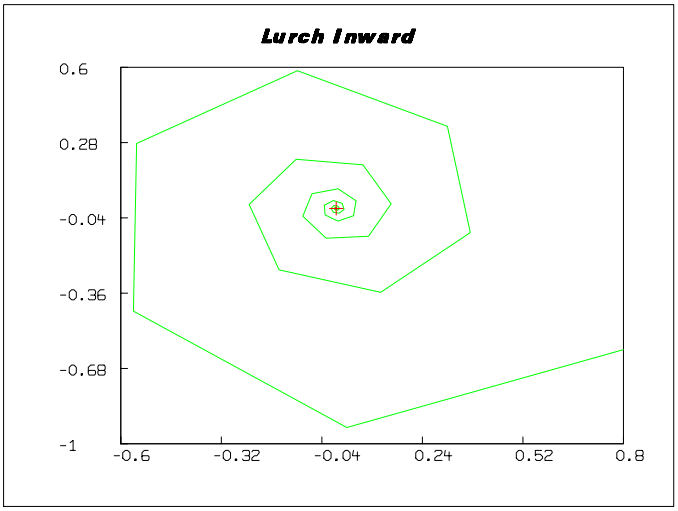

Figure B.13: Window `INBOUND`.

B.5 DIRKB.DO

The do-file DIRKB.DO defines a two-body orbit model using Hamilton's equations. DIRKB.DO is dedicated to the astronomer Dirk Brouwer who advocated Hamiltonian formalisms for celestial mechanics.

```
/* FILE: DIRKB.DO */
ECHODO = 3
RESET

/* TWO-BODY ORBIT MODEL USING HAMILTON'S EQUATIONS
 FILE DIRKB.DO IS DEDICATED TO THE ASTRONOMER
 DIRK BROUWER, WHO ADVOCATED HAMILTONIAN
 FORMALISMS FOR CELESTIAL MECHANICS.*/
FCT RSQX(X,Y,Z) = X^2 + Y^2 + Z^2
FCT RSQ(XDOT,YDOT,ZDOT) = XDOT^2 + YDOT^2 +ZDOT^2
FCT R(X,Y,Z) = SQRT(RSQX(X,Y,Z))
FCT H(X,Y,Z,XDOT,YDOT,ZDOT) = \
 RSQ(XDOT,YDOT,ZDOT)/2-1/R(X,Y,Z)
FCT XDOT'T(T) = -H'X(X,Y,Z,XDOT,YDOT,ZDOT)
FCT YDOT'T(T) = -H'Y(X,Y,Z,XDOT,YDOT,ZDOT)
FCT ZDOT'T(T) = -H'Z(X,Y,Z,XDOT,YDOT,ZDOT)
FCT X'T(T) = H'XDOT(X,Y,Z,XDOT,YDOT,ZDOT)
FCT Y'T(T) = H'YDOT(X,Y,Z,XDOT,YDOT,ZDOT)
FCT Z'T(T) = H'ZDOT(X,Y,Z,XDOT,YDOT,ZDOT)
/* THESE ARE HAMILTON'S CANONICAL EQUATIONS OF MOTION.*/
PAUSE

S = "M = INTEGRATE(X,Y,Z,XDOT,YDOT,ZDOT,T0:TMAX!NSTEP)"
INIT X(T0) = X0
INIT Y(T0) = Y0
INIT Z(T0) = Z0
INIT XDOT(T0) = XDOT0
INIT YDOT(T0) = YDOT0
INIT ZDOT(T0) = ZDOT0
X0=0.75; Y0=0; Z0=0; T0 = 0
XDOT0=0; RDOT0=1.290994448735806
EYE = 30.0; YDOT0 = RDOT0*SIND(EYE);
ZDOT0 = RDOT0*COSD(EYE)
TMAX=10; NSTEP=300
DO S
TOP TITLE "Keplerian Motion" FONT 21, AT (0, 0.90)
```

```
ORBIT="DRAW M COL(2,4) COLOR GREEN"
SUN="DRAW (O& 'O) LT NONE PT STAR COLOR ORANGE"
DO ORBIT; DO SUN;
KEPLER = W;
VIEW KEPLER;
```

Window KEPLER is shown in Figure B.14.

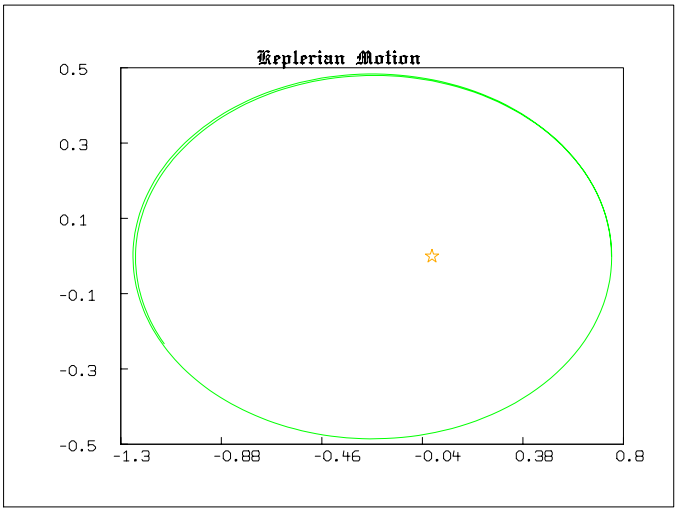

Figure B.14: Window KEPLER.

B.6 FILTER.DO

The do-file `FILTER.DO` defines a stable, linear ode with 3 distinct inputs. After the initial impulse dampens out, the solution with a cosine input becomes a sinusoidal output of different magnitude and shifted in phase. Linear odes act like filters, so they can provide interpretations of noisy and complex systems.

The model here is S.K. Jordan's 2^{nd} order Markov system,

$$\frac{d^2}{dt^2}u = f(t)$$

with $f = 0, \cos$, and noise.

```
/* FILE: FILTER.DO */
ECHODO = 3
RESET

/* THIS IS FILTER.DO IT IS A STABLE, LINEAR ODE WITH 3
 DISTINCT INPUTS. AFTER THE INITIAL IMPULSE DAMPENS
 OUT, THE SOLUTION WITH A COSINE INPUT BECOMES A
 SINUSOIDAL OUTPUT OF DIFFERENT MAGNITUDE AND SHIFTED
 IN PHASE. LINEAR ODES ACT LIKE FILTERS, SO THEY CAN
 PROVIDE INTERPRETATIONS OF NOISY AND COMPLEX SYSTEMS.
 THE MODEL HERE IS S. K. JORDAN'S 2ND-ORDER MARKOV
 SYSTEM: [(D/DT-B)^2]U = F(T),
 WITH F = 0, COS, AND NOISE. */

/* REFORMULATE ODE AS 1ST-ORDER: */
FCT Y'T(T) = A11*Y + A12*U + F(T)
FCT U'T(T) = Y
/* WE MUST HAVE B < 0.*/
B = -0.3
A11 = 2*B; A12 = -B*B
FCT F1(T) = 0.72*COS(A*T); A = 1.27
INITIAL U(0) = 0.8
INITIAL Y(0) = -0.6
SPIN = "N=INTEGRATE(U,Y,0:50!NSTEPS)"

/* CHOOSE NSTEPS = 2^N, N AN INTEGER, FOR FFTS.*/
NSTEPS = 512
FCT F(T) = 0; DO SPIN
```

```
TOP TITLE "Dampen Out" FONT 7
DRAW N COL(1,2) COLOR ORANGE LT DASHED
DRAW N COL(1,4) COLOR RED LT LDASH
DAMPEN=W
VIEW DAMPEN

BLANK DAMPEN
FCT F(T) = F1(T)
DO SPIN
N COL 6 = F ON (N COL 1)
DRAW N COL(1,2) COLOR ORANGE
DRAW N COL(1,6) COLOR GREEN LT LDASH
TOP TITLE "Dampen to a Cosine" FONT 7
FILTER = W
VIEW FILTER

BLANK FILTER
START=NORMRAN(EXP(1))
FCT F2(ARG)=NORMRAN(0)
M COL 1 = N COL 1
FCT F(T) = LOOKUP(M,T)
M COL 2 = F2 ON (M COL 1)
DO SPIN
TOP TITLE "2nd-Order Continuous Markov Process" FONT 7
DRAW N COL (1,2) COLOR ORANGE
STANLEY = W
VIEW STANLEY

BLANK STANLEY
AAA ROW 1 COL 1 = A11
AAA ROW 1 COL 2 = A12
AAA ROW 2 COL 1 = 1
AAA ROW 2 COL 2 = 0
EA = EIGEN(AAA)

/* HERE ARE THE EIGENVALUES FOR THE FILTER: */
TYPE EA COL 1 ROW(1,2)
TYPE EA COL 2 ROW(1,2)
DEL AAA

/* SELECT FINAL STATES; GET FFT: */
TS COL 1 = N COL 1
```

```
TS COL 2 = N COL 2
TS COL 3 = 0; DEL N
FFT = DFT(TS); DEL TS

/* GET POWER SPECTRUM = FFT*CONJ(FFT) */
FFT COL 4 = (FFT COL 2)*'(FFT COL 2) + \
 (FFT COL 3)*'(FFT COL 3)
DRAW FFT COL(1,4) COLOR GREEN
TOP TITLE "Power Spectrum Markov II" FONT 7
PSM2 = W
VIEW PSM2

BLANK PSM2
FFT COL 5 = LN ON (FFT COL 4)
DRAW FFT COL(1,5) COLOR GREEN
TOP TITLE "LN of Power Spectrum Markov II" FONT 7
LNPS2 = W
VIEW LNPS2

BLANK LNPS2
/* GET ACF FROM INVERSE FFT ON SPECTRUM: */
FFT COL 2 = FFT COL 4
DEL FFT COL(4:5)
FFT COL 3 = 0
ACF=IDFT(FFT)
ACFMAX = MAXV(ACF COL 2)
ACF COL 2 = (ACF COL 2)/ACFMAX
LONG=FLOOR(NROWS(ACF)/2)
DRAW ACF COL (1,2) ROW(1:LONG) COLOR GREEN
AXE COL (1:2) ROW (1:2) = 0
AXE COL 1 ROW 2 = 25
DRAW AXE COLOR RED
TOP TITLE "ACF FOR Markov II" FONT 7
ACFMARK2=W
VIEW ACFMARK2
```

Window `DAMPEN` is shown in Figure B.15.
Window `FILTER` is shown in Figure B.16.
Window `STANLEY` is shown in Figure B.17.
Window `PSM2` is shown in Figure B.18.
Window `LNPS2` is shown in Figure B.19.
Window `ACFMARK2` is shown in Figure B.20.

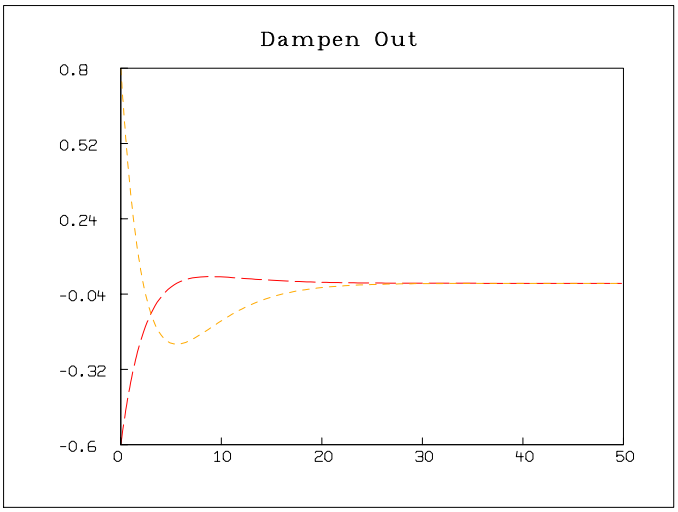

Figure B.15: Window `DAMPEN`.

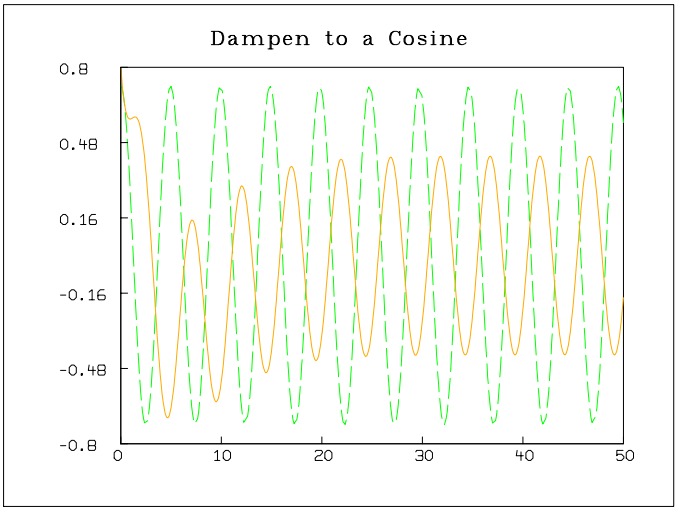

Figure B.16: Window `FILTER`.

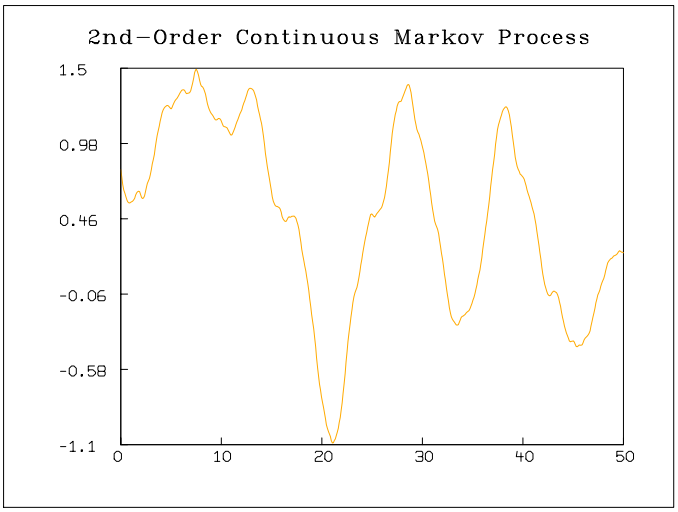

Figure B.17: Window `STANLEY`.

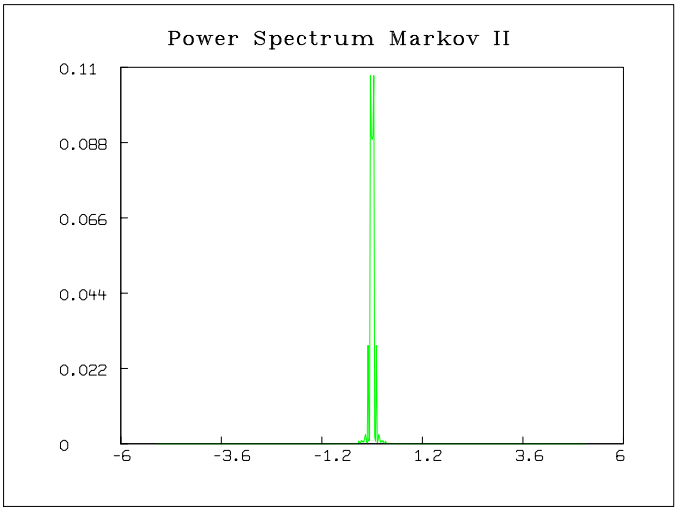

Figure B.18: Window `PSM2`.

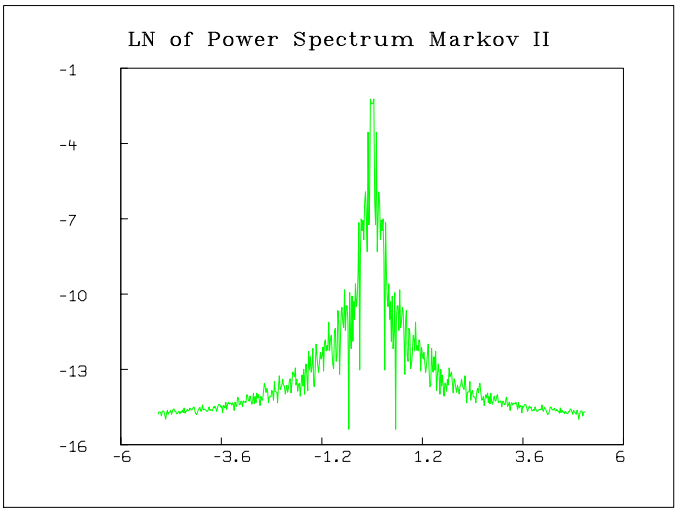

Figure B.19: Window `LNPS2`.

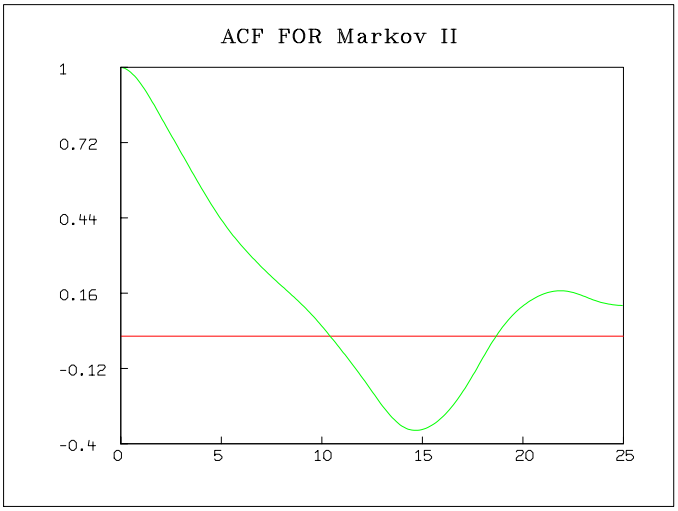

Figure B.20: Window `ACFMARK2`.

B.7 FITCYCLE.DO

The do-file `FITCYCLE.DO` demonstrates how random noise inputs affect the classic Volterra predator-prey equations.

```
/* FILE: FITCYCLE.DO */
ECHODO = 3
RESET

/* PREDATOR(Y)-PREY(X) MODEL
 THIS DEMO ILLUSTRATES HOW RANDOM NOISE INPUTS
 AFFECT THE CLASSIC VOLTERRA PREDATOR-PREY
 EQUATIONS. WHY ARE THE X AND Y FACTORS USED
 IN THE MODIFED ODES?
 (HINT: THINK LOGARITHMS.) */

R1 = "M COL 1 = T0:TF!NSTEPS"
R2 = "START = RAN(1001,-1,1)"
R3 = "M COL 2 = BOXCAR ON (M COL 1)"
R4 = "MM COL 1 = M COL 1"
R5 = "MM COL 2 = BOXCAR ON (MM COL 1)"
FCT BOXCAR(W) = RAN(0,-1,1)
S="SVECTR=INTEGRATE(X,Y,T0:TF!NSTEPS)"
FCT Y'T(T) = K*X*Y - C*Y + Y*LOOKUP(M,T)
FCT X'T(T) = A*X - B*X*Y + X*LOOKUP(MM,T)
T0=0; TF = 8; NSTEPS = 576
INIT X(0)=X0; INIT Y(0)=Y0; Y0=1; X0=.8
K=1;C=1;A=1;B=1

/* STORE THE RANDOM NUMBER MATRICES. */
DO R1; DO R2; DO R3; DO R4; DO R5

/* INTEGERATE THESE COUPLED, NONLINEAR,
 NOISE-DRIVEN ODES.*/
DO S
FCT CAPK(X,Y)=B*X-A*LN(B*X/A)+K*Y-C*LN(K*Y/C)

/* PLOT A TIME SERIES GRAPH OF BOTH
 VARIABLES:...*/
TOP TITLE "Predator-Prey with Noise" FONT 11
DRAW SVECTR COL (1,2) COLOR RED
DRAW SVECTR COL (1,4) COLOR GREEN
```

```
PPNZ1 = W
VIEW PPNZ1

BLANK PPNZ1
/* EVALUATE THE INTEGRAL ON THE NUMERICAL
 SOLUTION:...*/
SVXY COL 1 = SVECTR COL 2
SVXY COL 2 = SVECTR COL 4
KAPP = CAPK ON SVXY
SVXY COL 2 = KAPP
SVXY COL 1 = SVECTR COL 1
REFF(1,1)=T0; REFF(2,1)=TF
REFF(1,2)=CAPK(X0,Y0); REFF(2,2)=REFF(1,2)
DEL KAPP

/* FIDGET WITH THE GRAPH:...  */
DRAW SVXY COL(1,2) COLOR BROWN
DRAW REFF LT DOTTED COLOR RED
YAXIS FORMAT(-3,7,0,0,4,0) OFFSET(-0.15,-0.01) IN W
TOP TITLE "Volterra Integral Fluctuation" FONT 11
PPNZ2 = W
VIEW PPNZ2

BLANK PPNZ2
/* PLOT THE PHASE PLANE DIAGRAM ...
 WHICH SHOULD BE A CONTOUR OF THE INTEGRAL:...  */
DRAW SVECTR COL (2,4) COLOR BROWN
TOP TITLE "Phase Plane with Noise" FONT 11
PPNZ3 = W
VIEW PPNZ3

BLANK PPNZ3
/* REDEFINE ODES FOR NOISE-FREE MODEL: */
FCT Y'T(T)=K*X*Y-C*Y
FCT X'T(T)=A*X-B*X*Y

/* SELECT DATA FROM NOISE-DRIVEN SIMULATION.*/
NZ1 = SVECTR COL(1:2) ROW (7:NSTEPS:20)
NZ2 = SVECTR COL(1,4) ROW (7:NSTEPS:20)
CONSTRAINTS BIOLOGY= {X0>0,Y0>0,A>0,C>0,K>0,B>0 }
FIT(X0,Y0,A,B,C,K), X TO NZ1, Y TO NZ2, \
 CONSTRAINTS BIOLOGY
```

```
SVECTR = POINTS(X,Y,T0:TF!NSTEPS)
TOP TITLE "Fit to Noise-Driven System" FONT 11
DRAW SVECTR COL 2:3 COLOR BROWN
DRAW (NZ1 COL 2)& '(NZ2 COL 2) LT NONE \
 PT CIRCLE COLOR GREEN
PPNZ4 = W
VIEW PPNZ4
```

Window PPNZ1 is shown in Figure B.21.
Window PPNZ2 is shown in Figure B.22.
Window PPNZ3 is shown in Figure B.23.
Window PPNZ4 is shown in Figure B.24.

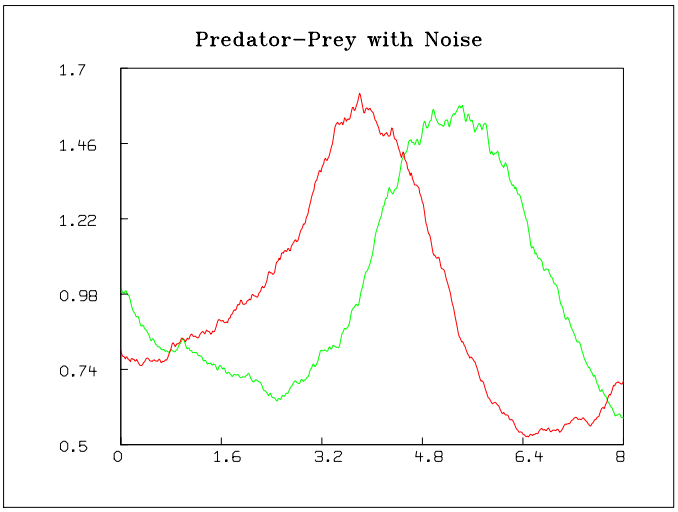

Figure B.21: Window `PPNZ1`.

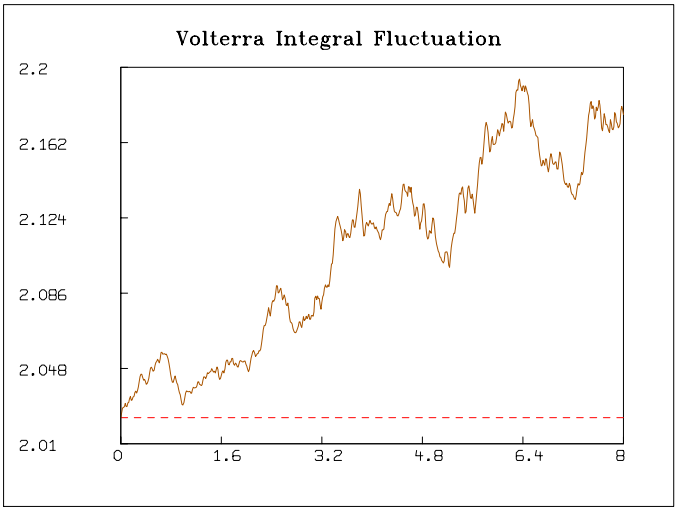

Figure B.22: Window `PPNZ2`..

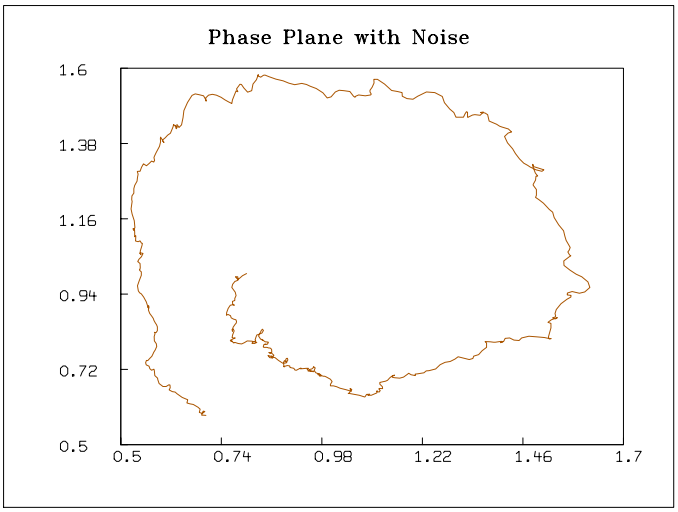

Figure B.23: Window `PPNZ3`.

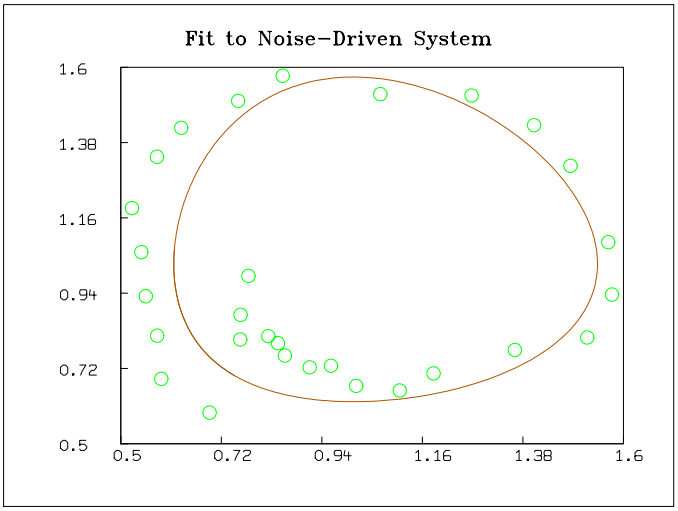

Figure B.24: Window `PPNZ4`.

B.8 HUBBERT.DO

The do-file HUBBERT.DO creates a model of the depletion of a nonrenewable resource and then fits the continuous logistic curve to it. This is the exceptional model where a differential equation can be solved in closed form. Even though this is one of the most trivial of solvable systems, it has proven adequate for its task. The late M. King Hubbert used it with a bit of insight to forecast the decline of U.S. domestic oil production in the mid-1970s. This program provides a simplified simulation of that history.

```
/* FILE: HUBBERT.DO */
ECHODO = 3
RESET

/* HUBBERT.DO CREATES A MODEL OF THE DEPLETION
 OF A NONRENEWABLE RESOURCE AND THEN FITS THE
 CONTINUOUS LOGISTIC CURVE TO IT.
 THIS IS THE EXCEPTIONAL MODEL WHERE A
 DIFFERENTIAL EQUATION CAN BE SOLVED IN CLOSED FORM.
 EVEN THOUGH THIS IS ONE OF THE MOST TRIVIAL OF SOLVABLE
 SYSTEMS, IT HAS PROVEN ADEQUATE FOR ITS TASK. THE LATE
 M. KING HUBBERT USED IT WITH A BIT OF INSIGHT TO
 FORECAST THE DECLINE OF U.S. DOMESTIC OIL PRODUCTION
 IN THE MID 1970S. THIS PROGRAM PROVIDES A SIMPLIFIED
 SIMULATION OF THAT HISTORY.*/

/* FIRST CREATE A MODEL OF THE OIL: */
PEAK = 1952; VARP = 32^2; RRATE = 1./10.
WELL COL 1 ROW 1 = NORMRAN(1013,PEAK,VARP)
NWELLS=50

/* DATE WELL OPENS: */
FOR I=1:NWELLS DO { WELL ROW I COL 1 = NORMRAN(0,PEAK,VARP) }

/* TOTAL RECOVERABLE CONTENTS: */
WELL COL 2 ROW 1 = RAN(1009,25,100)
FOR I=1:NWELLS DO { WELL ROW I COL 2 = RAN(0,25,100); }
ALLOFIT=ROWSUM(WELL COL 2)
TYPE ALLOFIT

/* EXPONENTIAL DRAINING: */
FCT EXXON(TAU,QUANT,TEE)=QUANT*(1.0-EXP[RRATE*(TAU-TEE)])
FCT PUMP(TAU,QUANT,TEE)=IF TEE>TAU THEN EXXON(TAU,QUANT,TEE) \
```

```
 ELSE 0
TYME ="WELL ROW (1:NWELLS) COL 3 = DATE"
BURN ="USED ROW(1:NWELLS) = PUMP ON WELL"
COUNT="GONE=ROWSUM(USED)"
GRR = "GRAF COL 1 ROW K = DATE; GRAF COL 2 ROW K = GONE"
REPORT="DO TYME; DO BURN; DO COUNT; DO GRR"
FOR K=1:10 DO { DATE = 1860+20*K; DO REPORT; }
GRAF COL 2 = 100.*(GRAF COL 2)/ALLOFIT(1)
DRAW GRAF COL(1,2) LT NONE PT CIRCLE COLOR BROWN
DRAW GRAF COL(1,2) COLOR GREEN

/* REVIEW THE SIMULATION:...*/
TOP TITLE "Total MKING1=W
VIEW MKING1

BLANK MKING1
/* DEFINE THE CONTINUOUS LOGISTIC:...*/
FCT DENO(A,C,X0,T,T0)=1.+(C/X0-1.)*EXP[-A*(T-T0)]
FCT LOGISTIC(A,C,X0,T,T0)=C/DENO(A,C,X0,T,T0)
C=100; T0=1890; X0=5; A= 0.042

/* DEFINE THE MODEL FOR THE FIT STATEMENT:...*/
CONSTRAINTS GEOLOGY = (C<500, C>20,A>0)
FCT LGFIT(T) = LOGISTIC(A,C,X0,T,T0)
FLOG="FIT (A,C,X0), LGFIT TO (GRAF ROW(1:J)) CONSTRAINTS GEOLOGY"
DPCT="DRAW GRAF COL(1,2) ROW(1:J) LT NONE PT CIRCLE COLOR BROWN"
A4CAST COL 1 = GRAF COL 1
FCST="A4CAST COL 2 = LGFIT ON (A4CAST COL 1)"
DPCT2="DRAW A4CAST COL (1,2) COLOR RED"
TOP TITLE "Oil Use Forecast 1" FONT 8
J=3; DO FLOG; DO DPCT; DO FCST; DO DPCT2;
MKING2 = W
VIEW MKING2

BLANK MKING2
J=4; DO FLOG; DO DPCT; DO FCST; DO DPCT2
TOP TITLE "Oil Use Forecast 2" FONT 8
MKING3 = W
VIEW MKING3

BLANK MKING3
J=5; DO FLOG; DO DPCT; DO FCST; DO DPCT2
```

```
TOP TITLE "Oil Use Forecast 3" FONT 8
MKING4 = W
VIEW MKING4

BLANK MKING4
J=10; DO FLOG; DO DPCT; DO FCST; DO DPCT2
TOP TITLE "Oil Use History" FONT 8
MKING5 = W
VIEW MKING5
```

Window MKING1 is shown in Figure B.25.
Window MKING2 is shown in Figure B.26.
Window MKING3 is shown in Figure B.27.
Window MKING4 is shown in Figure B.28.
Window MKING5 is shown in Figure B.29.

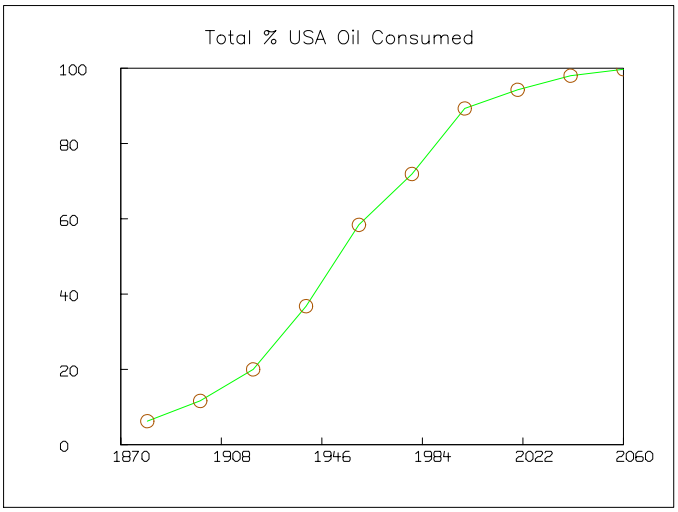

Figure B.25: Window `MKING1`.

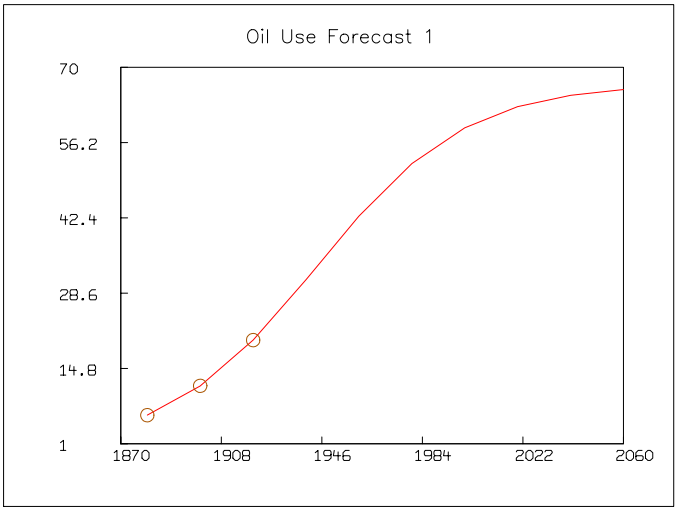

Figure B.26: Window `MKING2`.

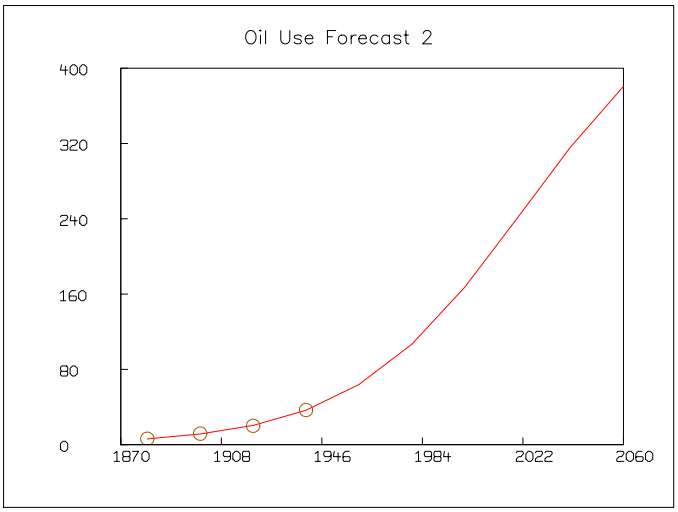

Figure B.27: Window `MKING3`.

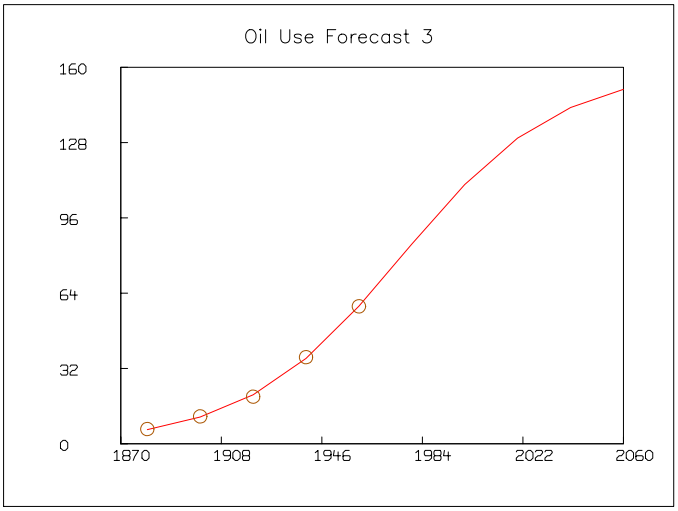

Figure B.28: Window `MKING4`.

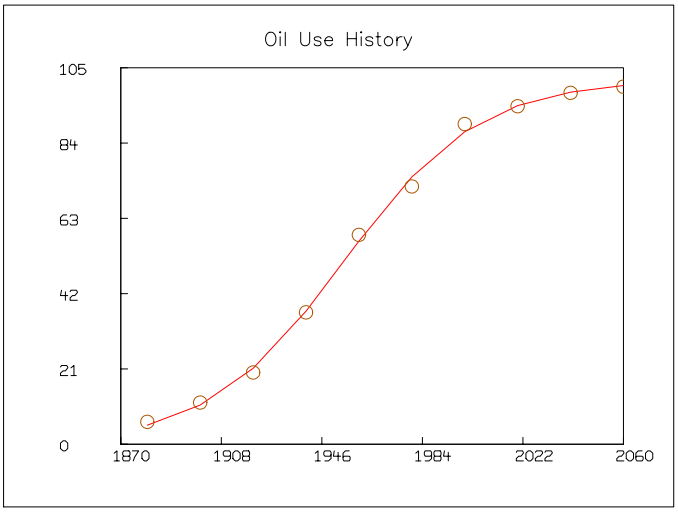

Figure B.29: Window `MKING5`.

B.9 KUGEL.DO

The do-file KUGEL.DO creates a graph of the Legendre polyomials of degrees 0, 1, 2, 3, 4, 5, 6, and 7.

```
/* FILE: KUGEL.DO */
ECHODO = 3
RESET

/* THIS KUGEL.DO FILE CREATES A GRAPH OF LEGENDRE
 POLYNOMIALS OF DEGREES 0 - 7 WITH GERMAN LABELS.
 CF. LEGENDRE.DO. */

/* SELECT A GERMAN TITLE IN GERMAN GOTHIC.*/
TOP TITLE "Die Kugelfunktionen Ordnung 0 - 7" FONT 22
TOP TITLE AT (0, 0.97)

/* DEFINE THE LIMITS OF THE WINDOW WHICH WILL HOLD
 THE GRAPH:*/
WINDOW 0.  TO 90., -1.  TO 1.1

/* CREATE THE ARGUMENTS LIST FOR THE LEGENDRE
 POLYNOMIALS */
NSTEPS = 301
LEGFC COL 1 = (0:90!NSTEPS)
ARGGS COL 1 = COSD ON (LEGFC COL 1)

/* GENERATE LEGENDRE POLYNOMIALS DEGREES 0-7 IN
 MATRIX LLX USING THE COSINE AS ARGUMENT. */
FOR I = 0:7 DO { \
 ARGGS COL 2 = I;
 LEGFC COL(I+2) = LEG ON ARGGS
  }

/* GRAPH THE LEGENDRE POLYNOMIALS IN DISTINCT COLORS:*/
DRAW LEGFC COL(1,2) COLOR GREEN
DRAW LEGFC COL(1,3) COLOR BROWN
DRAW LEGFC COL(1,4) COLOR RED
DRAW LEGFC COL(1,5) COLOR ORANGE
DRAW LEGFC COL(1,6) COLOR VIOLET
DRAW LEGFC COL(1,7) COLOR AQUA
DRAW LEGFC COL(1,8) COLOR ROSE
```

```
DRAW LEGFC COL(1,9) COLOR TURQUOISE

/* CF. JAHNKE & EMDE, TABLES OF FUNCTIONS (DOVER),
 P. 121.*/
KUGEL=W
VIEW KUGEL
```

Window KUGEL is shown in Figure B.30.

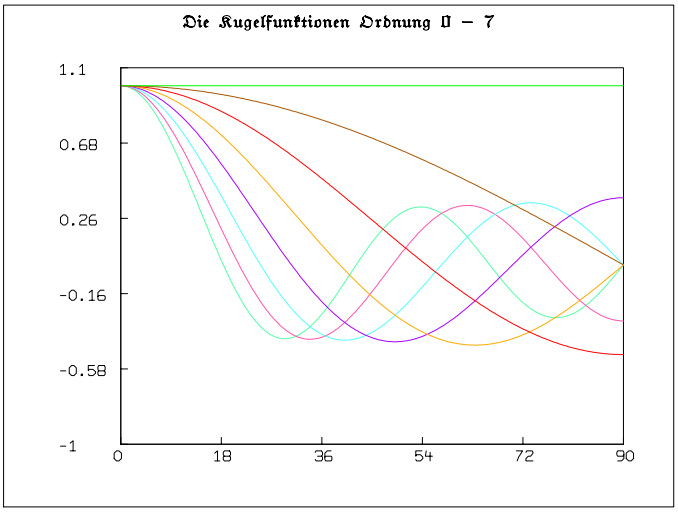

Figure B.30: Window KUGEL.

B.10 LALGEBRA.DO

The do-file LALGEBRA.DO illustrates a few properties of matrices that are important in dynamics. The eigenvalues of a square matrix of random numbers will usually imply instability, where it is the so-called system matrix of an ode or difference equation. This is useful for nonlinear systems, as well as linear ones; the demonstration is repeated for one such in LORENZ.DO B.14.

```
/* FILE: LALGEBRA.DO */
ECHODO = 3
RESET

PRAYER = "THESE NUMBERS SHOULD BE EQUAL!"
PRAYER2="THESE NUMBERS SHOULD BE SMALL IN MAGNITUDE!"

/* LALGEBRA.DO ILLUSTRATES A FEW PROPERTIES OF MATRICES
 THAT ARE IMPORTANT IN DYNAMICS. THE EIGENVALUES OF
 A SQUARE MATRIX OF RANDOM NUMBERS WILL USUALLY IMPLY
 INSTABILITY, WHERE IT IS THE SO-CALLED SYSTEM MATRIX
 OF AN ODE OR DIFFERENCE EQUATION. THIS IS USEFUL FOR
 NONLINEAR SYSTEMS, AS WELL AS LINEAR ONES; THE
 DEMONSTRATION IS REPEATED FOR ONE SUCH IN LORENZ.DO.*/

START=NORMRAN(SIN(1))
FCT RUN(X) = NORMRAN(0)
EIS = 7; KALT = EIS^2
A COL 1 = 1:KALT

TYPE " THIS VECTOR HAS LENGTH:"
NROWS(A)
A = RUN ON A
RMS = MNORM(A)/EIS;
MVL = ROWSUM(A)/KALT;
MEANVAL=MVL(1,1)
TYPE " THE MEAN & SIGMA SHOULD BE 0 & 1, BUT ARE:"
TYPE MEANVAL, RMS

/* REFORM INTO A SQUARE MATRIX:*/
A=SHAPE(EIS,EIS,A)
/* WITH DETERMINANT: */
DETA = DET(A); TYPE DETA;
PAUSE
```

```
/* NOW TEST THE INVERSE OF THE MATRIX:*/
AINV=A^-1
EYE=A*AINV
TYPE PRAYER; TYPE EIS; TYPE TRACE(EYE);
PAUSE

FOR I = 1:EIS DO { EYE(I,I) = EYE(I,I)-1; }
TYPE PRAYER2; MAXV(EYE); MINV(EYE);
PAUSE

/* GET THE EIGENVALUES AND PLOT IN THE COMPLEX
 PLANE: */
EAGER=EIGEN(A); EASY = EAGER'
TOP TITLE "Eigenvalues of a Matrix" FONT 20
DRAW EASY COL(1,2) LT NONE PT CIRCLE COLOR BROWN
IMAGE .24 TO .9, .15 TO .87 COLOR BLACK
DEL EASY COL(3:NCOLS(EASY))
SIDE1 = MAXV(EASY); SIDE2 = MINV(EASY)
EXTENT = MAX[ABS(SIDE1),ABS(SIDE2)]
DARK ROW (1:2) COL 1 = -EXTENT
DARK ROW (3:4) COL 1 = EXTENT
DARK ROW (1:3:2) COL 2 = EXTENT
DARK ROW (2:4:2) COL 2 = -EXTENT
DRAW DARK LT NONE PT DOTPT COLOR RED
THETA COL 1=0:(2*PI)!501
VROOM COL 1 = COS ON THETA
VROOM COL 2 = SIN ON THETA
DRAW VROOM LT DOTTED COLOR ORANGE
WAX COL 1 ROW(1:2)=0
WAX COL 2 =-EXTENT:EXTENT!2
DRAW WAX LT DASHED COLOR GREEN
LALG = W
VIEW LALG

BLANK LALG
/* SET UP A SINGULAR MATRIX:*/
ARNE = A; ARNE ROW(EIS)=0
BJ=ARNE^-1
ARNEBJ=BJ*ARNE
RTRIP=ARNE*ARNEBJ
LOSS=RTRIP-ARNE
TYPE PRAYER2, MAXV(LOSS), MINV(LOSS), DET(ARNE)
```

```
PAUSE
ZIGGY=EIGEN(ARNE)
SIGGY=ZIGGY'
TOP TITLE "Eigenvalues of Matrices" FONT 20
BOTTOM TITLE "...  Singular & Nonsingular" FONT 20
DRAW SIGGY COL(1:2) LT NONE PT SQUARE COLOR AQUA
SAME = W
VIEW SAME
```

Window `LALG` is shown in Figure B.31.
Window `SAME` is shown in Figure B.32.

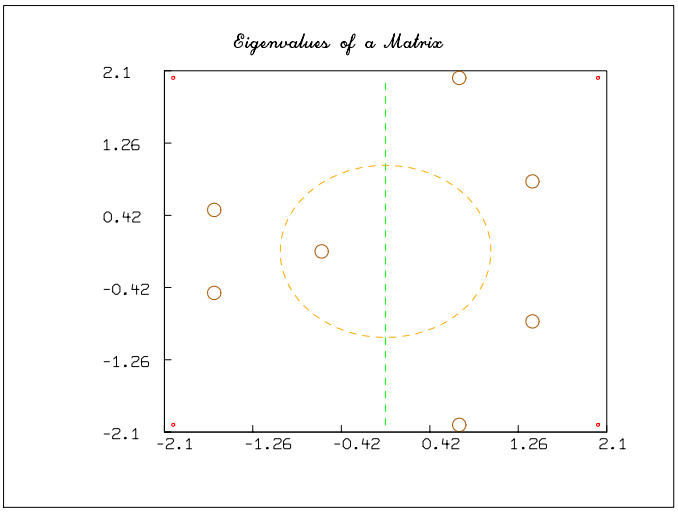

Figure B.31: Window `W`.

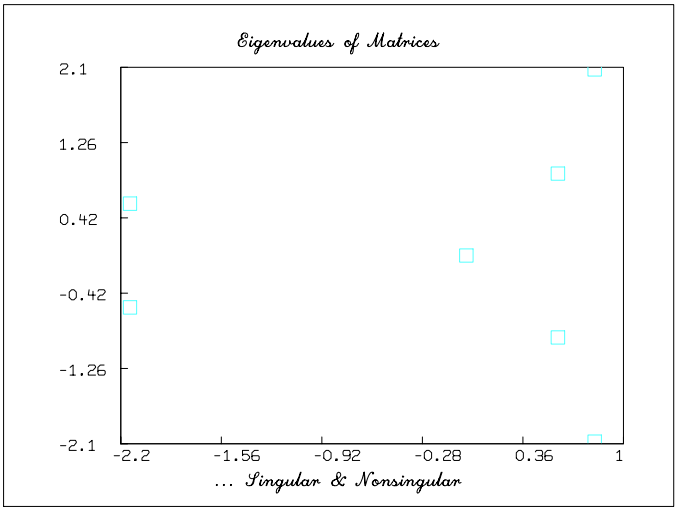

Figure B.32: Window `SAME`.

B.11 LEGENDRE.DO

The do-file `LEGENDRE.DO` creates a graph of Legendre polynomials of degrees 0,1,2,3,4,5, and 6.

```
/* FILE: LEGENDRE.DO */
ECHODO = 3
RESET

/* THIS LEGENDRE.DO FILE CREATES A GRAPH OF LEGENDRE
 POLYNOMIALS DEGREES 0 - 6.*/

/* SELECT A TITLE IN OLD ENGLISH: */
TOP TITLE "Legendre Polynomials Degrees 0 - 6" FONT 21
TOP TITLE AT (0, 0.97)

/* DEFINE THE LIMITS OF THE WINDOW WHICH WILL HOLD THE
 GRAPH:*/
WINDOW -1.  TO 1., -1.  TO 1.1

/* CREATE THE ARGUMENTS LIST FOR THE LEGENDRE
 POLYNOMIALS */
NSTEPS = 301
LEGFC COL 1 = (-1:1!NSTEPS); LLX COL 1 = LEGFC COL 1

/* GENERATE LEGENDRE POLYNOMIALS DEGREES 0-6 IN
 MATRIX LLX */
FOR I = 0:6 DO { \
 LEGFC COL 2 = I;
 LLX COL(I+2) = LEG ON LEGFC
  }

/* GRAPH THE LEGENDRE POLYNOMIALS IN DISTINCT COLORS: */
DRAW LLX COL(1,2) COLOR GREEN
DRAW LLX COL(1,3) COLOR BROWN
DRAW LLX COL(1,4) COLOR RED
DRAW LLX COL(1,5) COLOR ORANGE
DRAW LLX COL(1,6) COLOR VIOLET
DRAW LLX COL(1,7) COLOR AQUA
DRAW LLX COL(1,8) COLOR ROSE

LEGGS=W
VIEW LEGGS
```

Window `LEGGS` is shown in Figure B.33.

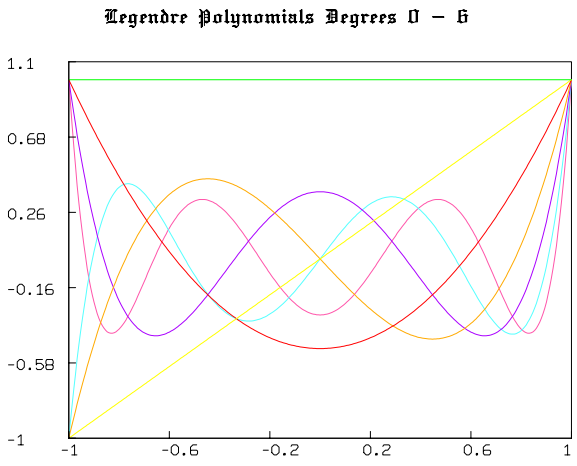

Figure B.33: Window LEGGS.

B.12 LIAPUNOV.DO

The do-file LIAPUNOV.DO investigates the famous Lorenz system by looking at a linearization at the equilibrium points. It can be said truly that if you have a dynamical model and do not know where its equilibrium points are, you do not know what you are doing. But this often is the case in much that is presented and published. In the case of Lorenz, it is easy to find where $\frac{dx_1}{dt}$, $\frac{dx_2}{dt}$, and $\frac{dx_3}{dt}$ all are 0, but in general, this will be difficult.

```
/* FILE: LIAPUNOV.DO */
ECHODO = 3
RESET

/* LIAPUNOV.DO INVESTIGATES THE FAMOUS LORENZ SYSTEM
 BY LOOKING AT A LINEARIZATION AT THE EQUILIBRIUM
 POINTS. IT CAN BE SAID TRULY THAT IF YOU HAVE A
 DYNAMICAL MODEL AND DO NOT KNOW WHERE ITS EQUILIBRIUM
 POINTS ARE, YOU DO NOT KNOW WHAT YOU ARE DOING. BUT
 THIS OFTEN IS THE CASE IN MUCH THAT IS PRESENTED
 AND PUBLISHED. IN THE CASE OF LORENZ, IT IS EASY TO
 FIND WHERE X1'T, X2'T, AND X3'T
 ALL ARE 0, BUT IN GENERAL THIS WILL BE DIFFICULT. */

/* THESE NONLINEAR ODES RESULT FROM THE SIMPLIFICATION
 OF A PDE.*/
FCT F1(X1,X2,X3) = A*(-X1+X2)
FCT F2(X1,X2,X3) = (B-X3)*X1 - X2
FCT F3(X1,X2,X3) = -C*X3 + X1*X2
"FCT X1'T(T) = F1(X1,X2,X3)"
"FCT X2'T(T) = F2(X1,X2,X3)"
"FCT X3'T(T) = F3(X1,X2,X3)"

/* THE CLASSIC PARAMETER SET IS: */
A=10; B = 28; C=8./3.

/* FORMULAS FOR COMPONENTS OF THE JACOBIAN
 MATRIX. THESE MAY BE APPLIED FOR ANY GIVEN
 F1, F2, F3.*/
FCT F11(X1,X2,X3) = F1'X1(X1,X2,X3)
FCT F12(X1,X2,X3) = F1'X2(X1,X2,X3)
FCT F13(X1,X2,X3) = F1'X3(X1,X2,X3)
FCT F21(X1,X2,X3) = F2'X1(X1,X2,X3)
FCT F22(X1,X2,X3) = F2'X2(X1,X2,X3)
```

```
FCT F23(X1,X2,X3) = F2'X3(X1,X2,X3)
FCT F31(X1,X2,X3) = F3'X1(X1,X2,X3)
FCT F32(X1,X2,X3) = F3'X2(X1,X2,X3)
FCT F33(X1,X2,X3) = F3'X3(X1,X2,X3)

/* TO FILL THE MATRIX:*/
S11 = "JACOBI(1,1) = F11(X1,X2,X3)"
S12 = "JACOBI(1,2) = F12(X1,X2,X3)"
S13 = "JACOBI(1,3) = F13(X1,X2,X3)"
S21 = "JACOBI(2,1) = F21(X1,X2,X3)"
S22 = "JACOBI(2,2) = F22(X1,X2,X3)"
S23 = "JACOBI(2,3) = F23(X1,X2,X3)"
S31 = "JACOBI(3,1) = F31(X1,X2,X3)"
S32 = "JACOBI(3,2) = F32(X1,X2,X3)"
S33 = "JACOBI(3,3) = F33(X1,X2,X3)"
S1 = "DO S11; DO S12; DO S13"
S2 = "DO S21; DO S22; DO S23"
S3 = "DO S31; DO S32; DO S33"
S = "DO S1; DO S2; DO S3"

/* FROM X1'T=0 WE GET X1=X2, SO THE EQ. PTS.
 ARE EASY TO FIND AS: */
ROOTS COL 1 ROW(1:3) = 0
BB1 = B-1
CBB1 = C*BB1
KAY = IF CBB1>=0 THEN 3 ELSE 1

/*...WHY?...*/
RTCB = SQRT[ABS(CBB1)]
ROOTS COL 2 ROW(1:2) = RTCB
ROOTS COL 3 ROW(1:2) = -RTCB
ROOTS COL(2:3) ROW 3 = BB1

/* THIS PART IS PROBLEM-DEPENDENT; FINDING THE
 EQ. PTS. IN SOME CASES MAY REQUIRE CONSIDERABLE
 EFFORT WITH FORMAL ANALYSIS AND MLAB COMPUTING.*/
LPV1="EG=EIGEN(JACOBI); EG=EG'; DEL EG COL(3:NCOLS(EG))"
PA = "SIDE1 = MAXV(EG); SIDE2 = MINV(EG)"
PB = "EXTENT = MAX[ABS(SIDE1),ABS(SIDE2)]"
PC = "DARK ROW (1:2) COL 1 = -EXTENT"
PD = "DARK ROW (3:4) COL 1 = EXTENT"
PE = "DARK ROW (1:3:2) COL 2 = EXTENT"
```

```
PF = "DARK ROW (2:4:2) COL 2 = -EXTENT"
PG = "DRAW DARK LT NONE PT DOTPT COLOR RED"
PH = "WAX COL 1 ROW(1:2) = 0; WAX COL 2 = -EXTENT:EXTENT!2"
PJ = "DRAW WAX LT DASHED COLOR GREEN"
PZ = "DO PA;DO PB;DO PC;DO PD;DO PE;DO PF;DO PG;DO PH;DO PJ"
LPV2 = "DO PZ;"
LIAEX = "L|PYNOV EXPONENTS"
TT = "TOP TITLE LIAEX FONT 24"
DRX = "DRAW EG COL(1,2) LT NONE PT CIRCLE COLOR BROWN"
LPV3 = "DO TT; DO DRX; VIEW W"
LPV4 = "LXSAVE = LXSAVE & 'EG"
LPV = "DO LPV1; DO LPV2; DO LPV3; DO LPV4"
LOAD = "X1 = ROOTS(1,J); X2 = ROOTS(2,J); X3 = ROOTS(3,J)"

GRAF1 = "W"; GRAF2= "=W"; G1="I"
AUGMNT = "GRAF1=GRAF1+G1"
GRAFOUT ="DO AUGMNT; GRAFX = GRAF1+GRAF2; DO GRAFX"

LXSAVE COL(1:2) ROW(1:3) = 0
FOR J=1:KAY DO { DO LOAD; DO S; DO LPV; DO GRAFOUT; }

TYPE" HERE ARE THE NUMBERS [IN LXSAVE COL(3:2*K+2):"
TYPE LXSAVE COL [3:(2*KAY+2)];
TYPE WINDOWS
```

Window `WI` is shown in Figure B.34.
Window `WII` is shown in Figure B.35.
Window `WIII` is shown in Figure B.36.

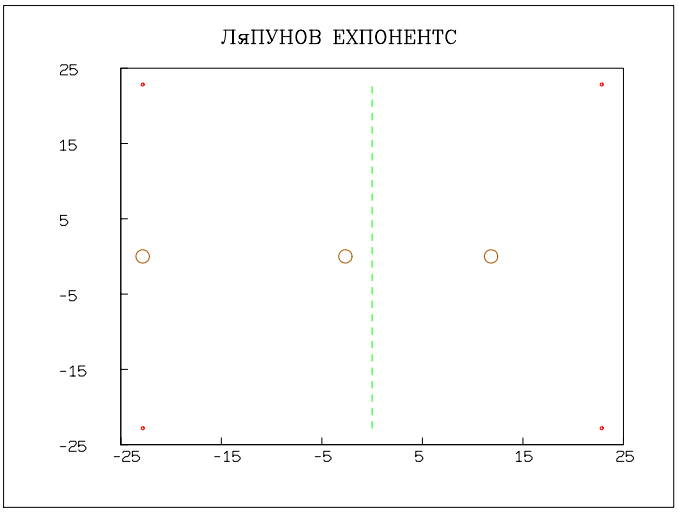

Figure B.34: Window `WI`.

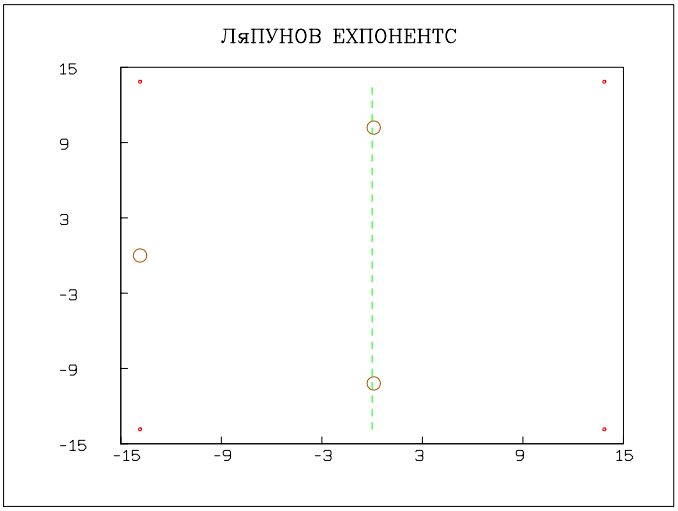

Figure B.35: Window `WII`.

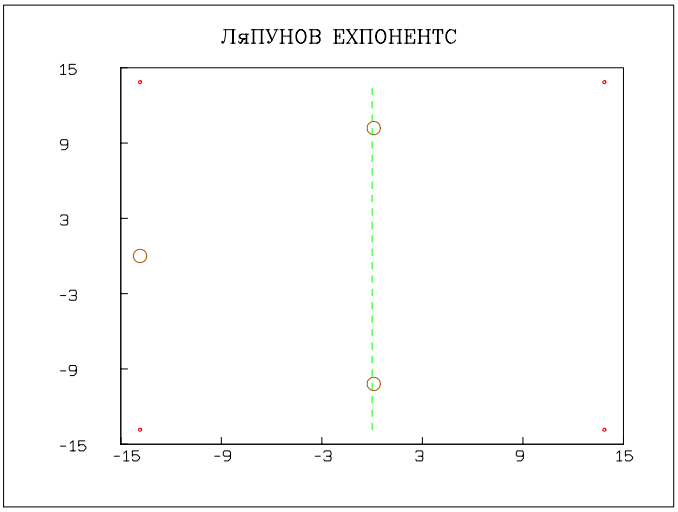

Figure B.36: Window `WIII`.

B.13 LINEAR.DO

The do-file LINEAR.DO finds a solution of a linear system of odes which spirals out and spirals in according to the defining constant coefficients. The same equation is integrated, but modified so that going forward in time is like going backward.

```
/* FILE: LINEAR.DO */
ECHODO = 3
RESET

/* THIS IS LINEAR.DO. IT SPIRALS OUT AND SPIRALS
 IN ACCORDING TO A LINEAR ODE WITH CONSTANT
 COEFFICIENTS. THE SAME EQUATION IS INTEGRATED,
 BUT MODIFED SO THAT GOING FORWARD IN TIME IS
 LIKE GOING BACKWARD.*/

/* LINEAR ODES, CONSTANT COEFFICIENTS;*/
FCT X1'T(T) = A(1,1)*X1 + A(1,2)*X2
FCT X2'T(T) = A(2,1)*X1 + A(2,2)*X2
A(1,1) = 27/250; A(1,2) = -107/125
A(2,1) = 143/125; A(2,2) = 24/125
INITIAL X1(0) = 0.8
INITIAL X2(0) = -0.6
SPIN = "N=INTEGRATE(X1,X2,0:50:0.1)"
DO SPIN

ORIGIN="DRAW (0& '0) LT NONE PT CROSSPT COLOR RED"
DO ORIGIN
DRAW N COL(2,4) COLOR BROWN
TOP TITLE "Spiral Outward" FONT 34
OUTWARD = W
VIEW OUTWARD

BLANK OUTWARD
LBL="EIGENVALUES OF THE SYSTEM MATRIX:"
EI="EA = EIGEN(A); TYPE LBL; TYPE EA COL (1:2) ROW(1,2)"
DO EI
A = -A
DO SPIN
DO ORIGIN
DRAW N COL(2,4) COLOR GREEN
TOP TITLE "Spiral Inward" FONT 34
```

```
INWARD = W
VIEW INWARD
```

The eigenvalues of the system matrix are:

```
       LBL =  EIGENVALUES OF THE SYSTEM MATRIX:

      : a  2 by 2 matrix

    1: -0.15         -0.15
    2: .988685997    -.988685997
```

Window `OUTWARD` is shown in Figure B.37.
Window `INWARD` is shown in Figure B.38.

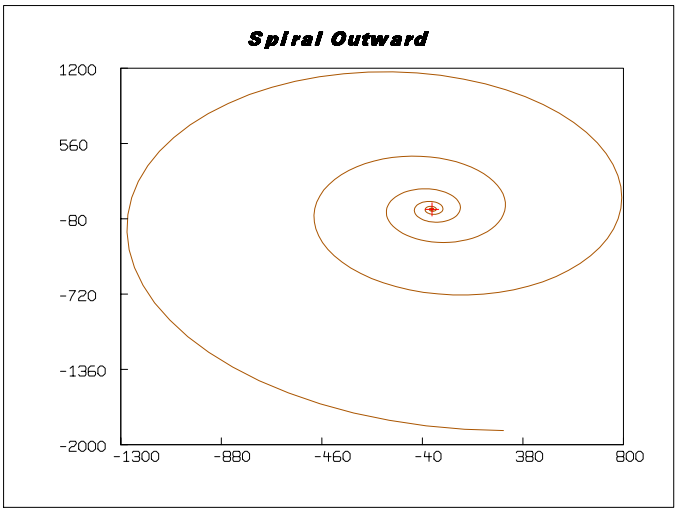

Figure B.37: Window `OUTWARD`.

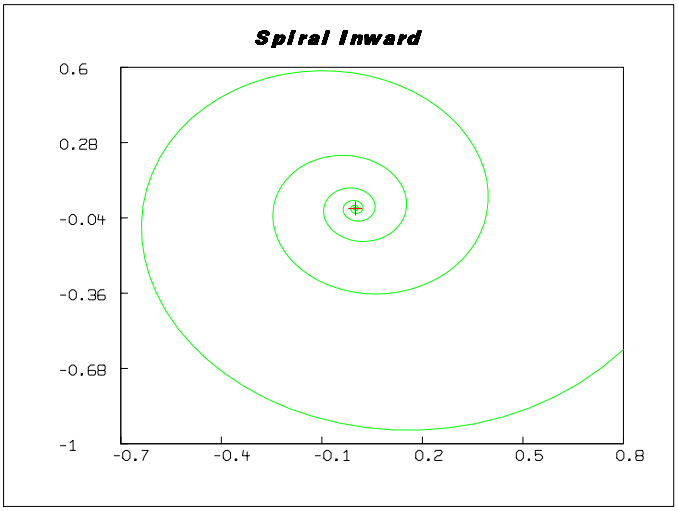

Figure B.38: Window `INWARD`.

B.14 LORENZ.DO

The do-file LORENZ.DO integrates and displays the famous Lorenz system. The three parameters, A,B, and C, chosen are the ones that generate the notrious butterfly. By using other parameters you display a variety of behaviors. .

```
/* FILE: LORENZ.DO */
ECHODO = 3
RESET

/* LORENZ.DO INTEGRATES AND DISPLAYS THE FAMOUS LORENZ
 SYSTEM. THE 3 PARAMETERS A, B, & C CHOSEN ARE THE
 ONES THAT GENERATE THE NOTORIOUS BUTTERYFLY. BY USING
 OTHER PARAMETERS YOU DISPLAY A VARIETY OF BEHAVIORS.*/

/* THESE NONLINEAR ODES RESULT FROM THE SIMPLIFICATION
 OF A PDE.*/
FCT F1(X1,X2,X3) = A*(-X1+X2)
FCT F2(X1,X2,X3) = (B-X3)*X1 - X2
FCT F3(X1,X2,X3) = -C*X3 + X1*X2
FCT X1'T(T) = F1(X1,X2,X3)
FCT X2'T(T) = F2(X1,X2,X3)
FCT X3'T(T) = F3(X1,X2,X3)
S = "M2 = POINTS(X1,X2,X3,M)"
INIT X1(T0) = X10
INIT X2(T0) = X20
INIT X3(T0) = X30

X10 = .01; X20 = .01; X30 = .01; T0 = 0
TMAX = 35; NSTEP = 2000
A = 10; B = 28; C = 8./3.
M COL 1 = T0:TMAX!NSTEP
DO S
SPAN = TMAX - T0
MX = SQRT(0.5)
M2 COL 1 = MX*(M2 COL 2 + M2 COL 3)
M2 COL 2 = M2 COL 4

/* DELETE M COL(3:7) */
TOP TITLE "The Lorenz Butterfly" FONT 8
DRAW M2 COL(1,2) COLOR GREEN
DRAW (0& '0) LT NONE PT CIRCLE
```

```
DRAW (12& '27) LT NONE PT CIRCLE
DRAW (-12& '27) LT NONE PT CIRCLE
BIGED = W
VIEW BIGED
```

Window `BIGED` is shown in Figure B.39.

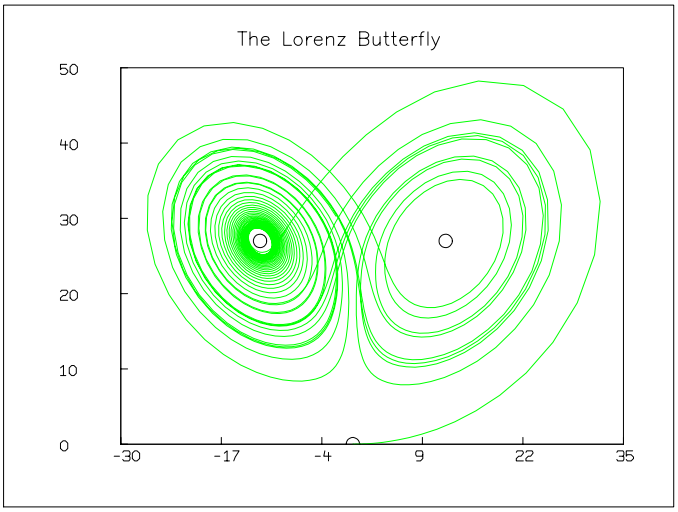

Figure B.39: Window `BIGED`.

B.15 MITCH.DO

The do-file MITCH.DO displays graphically the solutions to the discrete logistic equation. Unlike the continuous logistic—often called Hubbert's Pimple—the discrete logistic displays some of the very complex behavior typical of nonlinear systems. This equation was used by Prof. Mitchell Feigenbaum to launch the present era of interest in nonlinear dynamics, starting in 1975. Enter values of the parameter A from 0 to 4, and a number of steps (X0), when requested.

```
/* FILE: MITCH.DO */
ECHODO = 3
RESET

/* MITCH.DO DISPLAYS GRAPHICALLY THE SOLUTIONS TO THE
 DISCRETE LOGISTIC EQUATION. UNLIKE THE CONTINUOUS
 LOGISTIC -- OFTEN CALLED HUBBERT'S PIMPLE -- THE
 DISCRETE LOGISTIC DISPLAYS SOME OF THE VERY COMPLEX
 BEHAVIOR TYPICAL OF NONLINEAR SYSTEMS. THIS
 EQUATION WAS USED BY PROF. MITCHELL FEIGENBAUM TO
 LAUNCH THE PRESENT ERA OF INTEREST IN NONLINEAR
 DYNAMICS, STARTING IN 1975.  ENTER VALUES OF THE
 PARAMETER A FROM 0 TO 4 AND A NUMBER OF STEPS,
 WHEN REQUESTED.*/
A = 2.0; NSTOP=25; X0 = 0.5
TYPE " ENTER PARAMETER 0 <= A <= 4."
A = KREAD()
TYPE " ENTER INITIAL VALUE 0 <= X0 <= 1."
X0 = KREAD()

/* CONFINE A AND X0 TO SAFE VALUES. WHY???  */
A = IF A >= 0 THEN A ELSE 0
A = IF A <= 4 THEN A ELSE 4
X0 = IF X0 >= 0 THEN X0 ELSE 0
X0 = IF X0 <= 1 THEN X0 ELSE 1

/* MAKE LONGER RUNS LARGER A. WHY???  */
NSTOP = 5 + FLOOR(1.5*A*A)

/* DEFINE AND SOLVE THE DISCRETE LOGISTIC: */
FCT X 'N(N) = A*X*(1.0 - X)
INIT X(0)=X0
FBAUM=ITERATE(X 'N,NSTOP+1)
```

```
/* WHAT IS THIS PARABOLA? */
FCT Y(X)=A*X*(1.0 - X)

/* WHAT IS THIS LINE? */
FCT IDENT(X) = X

/* PLOT THE SOLUTION, THE PARABOLA AND THE LINE. */
ENVELOP COL 1 = 0:1!250
ENVELOP COL 2 = Y ON ENVELOP COL 1
DRAW ENVELOP COL (1,2) COLOR RED
PEAK = Y(0.5)
NPLOTZ = IF PEAK < 0.5 THEN FLOOR(250*PEAK + 5) \
 ELSE 250
LYNE COL 1 ROW 1:NPLOTZ = ENVELOP COL 1 ROW 1:NPLOTZ
LYNE COL 2 = IDENT ON LYNE COL 1
DRAW LYNE COL (1,2) COLOR GREEN
FCT FROW(I) = FLOOR(0.51 +I/2)
NST2 = 2*NSTOP + 2
FOR I=1:NST2 DO { FB COL 1 ROW I=FBAUM COL 2 ROW FROW(I); }
FOR I=3:NST2 DO { FB COL 2 ROW(I-1) = FB COL 1 ROW I; }
FB ROW 1 COL 2 = 0
FB COL 2 ROW NST2 = FB COL 1 ROW NST2
DRAW FB COL (1,2) COLOR BROWN
TOP TITLE "Discrete Logistic" FONT 16

IMAGE 0.2 TO 0.95, 0.14 TO 0.9 FFRACT
DEL ENVELOP; DEL LYNE; DEL FBAUM; DEL FB
WMITCH = W
VIEW WMITCH

TYPE " RENAME AND SAVE WMITCH TO PRESERVE "
TYPE " YOUR GRAPH. DELETE WMITCH, IF NOT "
TYPE " RENAMED, AND RERUN MITCH.DO TO SEE "
TYPE " MORE EXAMPLES.";
```

With `A = 2` and `X0 = .2` the resulting window `WMITCH` is shown in Figure B.40.

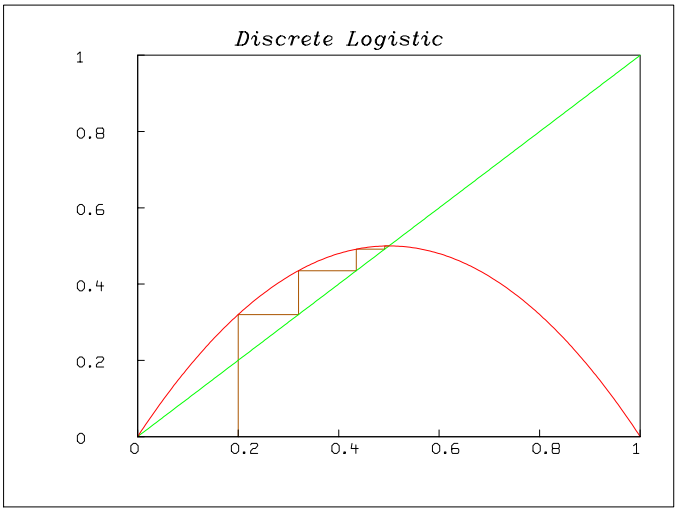

Figure B.40: Window `WMITCH`.

B.16 ORTHO.DO

The do-file `ORTHO.DO` does a Gram-Schmidt orthogonalization on the discretized interval [-1,+1] to illustrate:

1. orthogonality depends on domain definition.
2. Gram-Schmidt is sensitive to rounding error.

```
/* FILE: ORTHO.DO */
ECHODO = 3
RESET

/* FILE ORTHO.DO DOES GRAM-SCHMIDT ORTHOGONALIZATION
 ON THE DISCRETIZED INTERVAL [-1,+1] TO ILLUSTRATE
 1.  ORTHOGONALITY DEPENDS ON DOMAIN DEFINITION.
 2.  G-S IS SENSITIVE TO ROUNDING ERROR.*/

/* CREATE THE DISCRETE DOMAIN: */
NSTEPS = 41

/* CREATE 0 AND 1ST POWER COLUMNS.*/
X COL 1 ROW (1:NSTEPS) = 1
X COL 2 = (-1:+1!NSTEPS)

/* CREATE 2^N POWERS OF ARGUMENTS: */
FOR I = 1:3 DO { I1 = 2^I; I2 = I1/2; I1 = I1+1; I2 = I2+1;
X COL I1 = (X COL I2)*'(X COL I2); }

/* FILL IN THE REST:*/
X COL 4 = (X COL 2)*'(X COL 3)
X COL 6 = (X COL 5)*'(X COL 2)
X COL 7 = (X COL 5)*'(X COL 3)
X COL 8 = (X COL 5)*'(X COL 4)
X COL 10 = (X COL 9)*'(X COL 2)
X COL 11 = (X COL 9)*'(X COL 3)

IQ = 11
/* DO THE GRAM-SCHMIDT PROCEDURE:*/
DOTDOT="DD=[DOT(X COL I,XX COL J)]/[DOT(XX COL J,XX COL J)]"
GS1 = "XX COL I = XX COL I - DD*XX COL J"
```

```
GSS = "DO DOTDOT; DO GS1"
GSPROCESS = "XX=X; FOR I=2:IQ DO { FOR J=1:(I-1) DO { DO GSS; }; }"
DO GSPROCESS
IQQ = IQ+1;XX COL(IQQ) = X COL 2; DEL X

/* MAKE A MATRIX OF ALL INNER PRODUCTS:*/
DOTTY = "MXX(I,J) = DOT(XX COL I, XX COL J); MXX(J,I) = MXX(I,J)"
DOTPROCESS = "FOR I=1:IQ DO { FOR J=I:IQ DO { DO DOTTY; } }"
DO DOTPROCESS
TYPE TRACE(MXX)
PAUSE

/* NORMALIZE THE COLUMNS OF MATRIX XX:*/
NPROCESS = "FOR I=1:IQ DO { NXX=SQRT(MXX[I,I]);XX COL I=(XX COL I)/NXX }"
DO NPROCESS
DO DOTPROCESS
TYPE TRACE(MXX)
PAUSE

/* GET RMS OF MXX-I:*/
FOR I = 1:IQ DO { MXX(I,I)=MXX(I,I)-1; }
GOODNESS = MNORM(MXX,0)/IQ; TYPE GOODNESS
TYPE " SAVE THIS VALUE TO COMPARE WITH ORTHO2.DO."
PAUSE

TOP TITLE "Orthogonality on a Discrete Set" FONT 7
BOTTOM TITLE "...  from powers" FONT 7
DRAW XX COL (IQQ,1) COLOR RED LT DASHED
DRAW XX COL (IQQ,2) COLOR ORANGE LT DASHED
DRAW XX COL (IQQ,3) COLOR BROWN LT DASHED
DRAW XX COL (IQQ,4) COLOR GREEN LT DASHED
DRAW XX COL (IQQ,5) COLOR BLUE LT DASHED
DRAW XX COL (IQQ,6) COLOR PURPLE LT DASHED
DRAW XX COL (IQQ,7) COLOR RED
DRAW XX COL (IQQ,8) COLOR ORANGE
DRAW XX COL (IQQ,9) COLOR BROWN
DRAW XX COL (IQQ,10) COLOR GREEN
DRAW XX COL (IQQ,11) COLOR BLUE
SAM = MINV[XX COL(1:IQ)]
XX COL (IQQ+1) = 0.98*SAM
DRAW XX COL (IQQ,IQQ+1) COLOR ORANGE LT NONE PT UTICK
GSCHMIDT = W
```

```
VIEW GSCHMIDT
```

Window `GSCHMIDT` is shown in Figure B.41.

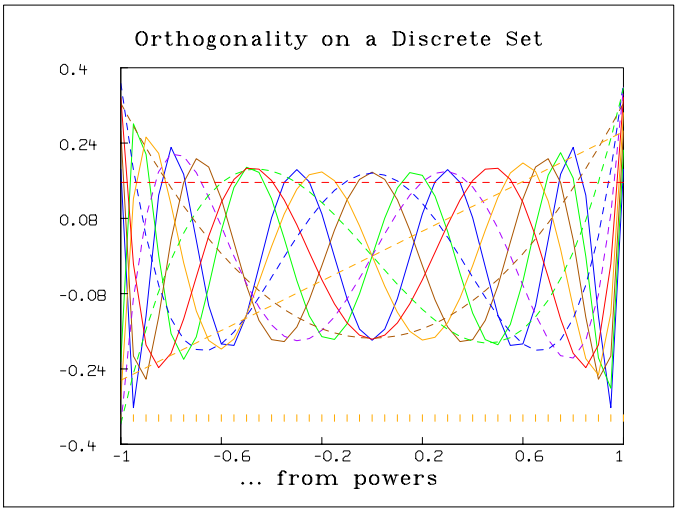

Figure B.41: Window `GSCHMIDT`.

B.17 ORTHO2.DO

The do-file `ORTHO2.DO` does the Gram-Schmidt orthogonalization on the discretized interval [-1,+1] to illustrate:

1. orthogonality depends on domain definition.
2. Gram-Schmidt is sensitive to rounding error.

By starting with Legendre polynomials, not powers, this computation attempts to reduce rounding errors. CF. The value of goodness with that from `ORTHO.DO` B.16.

```
/* FILE: ORTHO2.DO */
ECHODO = 3
RESET

/* FILE ORTHO2.DO DOES GRAM-SCHMIDT ORTHOGONALIZATION ON
 THE DISCRETIZED INTERVAL [-1, +1] TO ILLUSTRATE
 1.  ORTHOGONALITY DEPENDS ON DOMAIN DEFINITION.
 2.  G-S IS SENSITIVE TO ROUNDING ERROR.
 BY STARTING WITH LEGENDRE POLYNOMIALS, NOT POWERS, THIS
 COMPUTATION ATTEMPTS TO REDUCE ROUNDING ERRORS. CF. THE
 VALUE OF GOODNESS WITH THAT FROM ORTHO.DO.*/

/* CREATE THE DISCRETE DOMAIN:*/
NSTEPS = 41

/* CREATE 0 AND 1ST POWER COLUMNS.*/
DOMAIN COL 1 = (-1:+1!NSTEPS)

/* CREATE TABLE OF LEGENDRE POLYNOMIALS:*/
IQ = 11
FOR I=1:IQ DO { DOMAIN COL 2=(I-1); X COL I=LEG ON DOMAIN; }

/* DO THE GRAM-SCHMIDT PROCEDURE: */
DOTDOT = "DD=[DOT(X COL I,XX COL J)]/[DOT(XX COL J,XX COL J)]"
GS1 = "XX COL I = XX COL I - DD*XX COL J"
GSS = "DO DOTDOT; DO GS1"
GSPROCESS = "XX=X; FOR I=2:IQ DO { FOR J=1:(I-1) DO { DO GSS; } ; }"
DO GSPROCESS
IQQ = IQ+1
```

```
XX COL(IQQ) = DOMAIN COL 1
DEL X

/* MAKE A MATRIX OF ALL INNER PRODUCTS: */
DOTTY="MXX(I,J)=DOT(XX COL I, XX COL J); MXX(J,I)=MXX(I,J)"
DOTPROCESS = "FOR I=1:IQ DO { FOR J=I:IQ DO { DO DOTTY; } }"
DO DOTPROCESS
TYPE TRACE(MXX)
PAUSE

/* NORMALIZE THE COLUMNS OF MATRIX XX:*/
NPROCESS="FOR I=1:IQ DO { NXX=SQRT[MXX(I,I)];XX COL I=(XX COL I)/NXX; }"
DO NPROCESS
DO DOTPROCESS
TYPE TRACE(MXX)
PAUSE

/* GET RMS OF MXX-I: */
FOR I = 1:IQ DO { MXX(I,I)=MXX(I,I)-1; }
GOODNESS = MNORM(MXX,0)/IQ;TYPE GOODNESS

TYPE " SAVE THIS VALUE TO COMPARE WITH ORTHO.DO."

TOP TITLE "Orthogonality on a Discrete Set" FONT 7
BOTTOM TITLE "...  from Legendre Polynomials" FONT 7
DRAW XX COL (IQQ,1) COLOR RED LT DASHED
DRAW XX COL (IQQ,2) COLOR ORANGE LT DASHED
DRAW XX COL (IQQ,3) COLOR BROWN LT DASHED
DRAW XX COL (IQQ,4) COLOR GREEN LT DASHED
DRAW XX COL (IQQ,5) COLOR BLUE LT DASHED
DRAW XX COL (IQQ,6) COLOR PURPLE LT DASHED
DRAW XX COL (IQQ,7) COLOR RED
DRAW XX COL (IQQ,8) COLOR ORANGE
DRAW XX COL (IQQ,9) COLOR BROWN
DRAW XX COL (IQQ,10) COLOR GREEN
DRAW XX COL (IQQ,11) COLOR BLUE
SAM = MINV[XX COL(1:IQ)]
XX COL (IQQ+1) = 0.98*SAM
DRAW XX COL (IQQ,IQQ+1) COLOR ORANGE LT NONE PT UTICK
LEGDISCR = W
VIEW LEGDISCR
```

Window `LEGDISCR` is shown in Figure B.42.

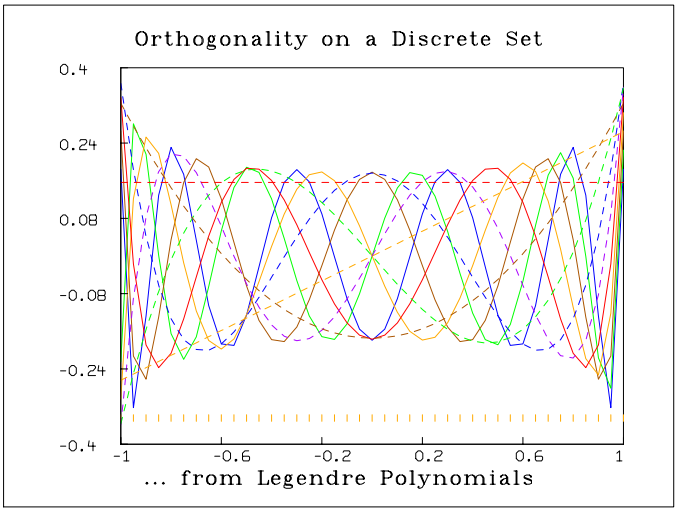

Figure B.42: Window `LEGDISCR`.

B.18 POLLY.DO

The do-file POLLY.DO computes Legendre polynomials on the domain [-1,+1] using the built-in MLAB function LEG and direct polynomial evaluation. The results are then compared and the differences plotted. After doing the 5^{th} order you can generate any other by typing in that order; enter 0 to stop.

```
/* FILE: POLLY.DO */
ECHODO = 3
RESET

/* POLLY.DO COMPUTES LEGENDRE POLYNOMIALS ON THE
 DOMAIN [-1,+1] USING THE INTERNAL FUNCTION LEG()
 AND DIRECT POLYNOMIALS EVALUATION. THE RESULTS
 ARE THEN COMPARED AND THE DIFFERENCES PLOTTED.
 AFTER DOING THE 5TH ORDER YOU CAN GENERATE ANY
 OTHER BY TYPING IN THAT ORDER; ENTER A 0 TO STOP.*/
NSTEPS = 301

SQUEEZE="YAXIS FORMAT(-3,4,0,0,4,0) OFFSET(-.14,-.01) IN W"
TORX = " Rounding Errors Legendre Polynomial"
TRIM = "TOP TITLE TORX FONT 9"
FCT ALFA(N) = IF N <= 1 THEN 1 ELSE ((N+N-1)/N)*ALFA(N-1)
FCT FAX(N,M) = -[(M+1)*(M+2)]/[(N-M)*(N+M+1)]
FCT VETA(N,M) = IF M>=(N-1) THEN 1 ELSE FAX(N,M)*VETA(N,M+2)
FCT LGFC(N,M) = ALFA(N)*VETA(N,M)
FCT PWR(XX,N,I)=IF ABS(XX)>0 THEN XX^(N-2*I) ELSE 0
FCT LP(XX,N) = SUM[I,0,FLOOR(N/2),LGFC(N,N-2*I)*PWR(XX,N,I)]
TYPE " THESE TWO NUMBERS SHOULD BE EQUAL:"
TYPE LP(.5,5)
TYPE LEG(.5,5)
PAUSE

FCT RNDERR(XX,N) = LP(XX,N)-LEG(XX,N)
/* CREATE A ROW OF VALUES -1 TO +1 FOR X */
Y COL 1 = (-1:1!NSTEPS); Y COL 2 = 5
S1 = "ROUNDER = RNDERR ON Y"
S2 = "Y COL 2 = ROUNDER"
S3 = "DRAW Y COL(1,2) COLOR BROWN; DO TRIM"
S4 = "ERROR = W; VIEW ERROR"
DO S1; DO S2; DO S3; DO SQUEEZE; DO S4
SS5 = "TYPE IN AN INTEGER OR 0 TO STOP:"
```

```
S5 = "TYPE SS5; NERD=KREAD()"
S6 = "NN= FLOOR(ABS(NERD))"
S8 = "Y COL 2 = NN"
FOR I = 1:33 DO { \
 DO S5; DO S6;
 IF NN<1 OR NN=MAXPOS THEN BREAK;
 DO S8; DO S1; DO S2;
 DEL ERROR; DO S3;
 DO SQUEEZE; DO S4; }
TYPE "YOUR LAST GRAPH IS IN:"; TYPE WINDOWS;
```

Window `ERROR`, resulting from the first iteration of `polly.do`, is shown in Figure B.43.

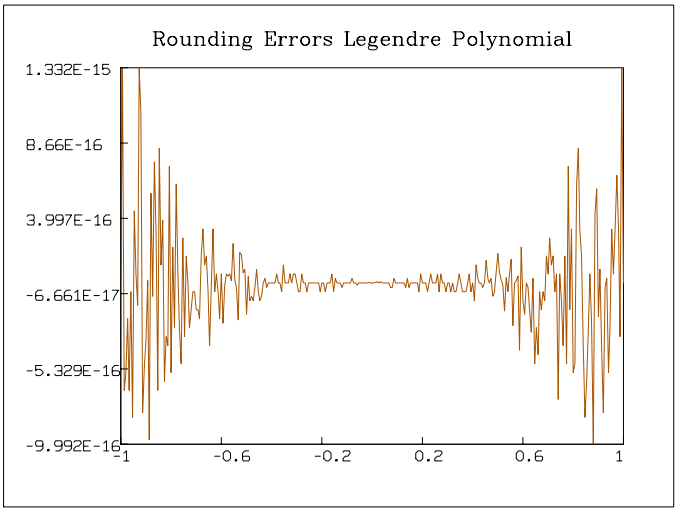

Figure B.43: Window ERROR.

B.19 POWER.DO

The do-file POWER.DO plots some positive integral powers on the domain [-1,+1]. This is an easy exercise and illustrates how poor polynomials are for representing functions; they are too much alike except for powers 0, 1, 2, and 3. Legendre polynomials, which are orthogonal on this domain are much more effective. Taylor and Laurent series are useful in formal analysis, where rounding errors are not a factor; for numerical work these power series are useful only on very restricted domains.

```
/* FILE: POWER.DO */
ECHODO = 3
RESET

/* POWER.DO PLOTS SOME POSITIVE INTEGRAL POWERS ON
 THE DOMAIN [-1,+1].  THIS IS AN EASY EXERCISE AND
 ILLUSTRATES HOW POOR POLYNOMIALS ARE FOR REPRESENTING
 FUNCTIONS; THEY ARE TOO MUCH ALIKE EXCEPT FOR POWERS
 0,1,2, AND 3.  LEGENDRE POLYNOMIALS, WHICH ARE
 ORTHOGONAL ON THIS DOMAIN ARE MUCH MORE EFFECTIVE.
 TAYLOR AND LAURENT SERIES ARE USEFUL IN FORMAL
 ANALYSIS, WHERE ROUNDING ERRORS ARE NOT A FACTOR;
 FOR NUMERICAL WORK THESE POWER SERIES ARE USEFUL
 ONLY ON VERY RESTRICTED DOMAINS.*/
NSTEPS = 301

/* CREATE A COLUMN OF 1S FOR X^0:  */
X COL 1 ROW(1:NSTEPS) = 1

/* CREATE A ROW OF VALUES -1 TO +1 FOR X */
X COL 2 = (-1:1!NSTEPS)

/* CREATE X^N */
FOR I=3:10 DO { X COL I = (X COL 2)*'(X COL (I-1)); }
BOTTOM TITLE "Graph of Powers 2-9" FONT 9
DRAW X COL (2,3)
DRAW X COL (2,4) COLOR BROWN
DRAW X COL (2,5) COLOR ROSE
DRAW X COL (2,6) COLOR GREEN
DRAW X COL (2,7) COLOR RED
DRAW X COL (2,8) COLOR AQUA
DRAW X COL (2,9) COLOR ROSE LT DASHED
DRAW X COL (2,10) COLOR BROWN LT DASHED
```

```
POTENZEN = W
VIEW POTENZEN
```

Window `POTENZEN` is shown in Figure B.44.

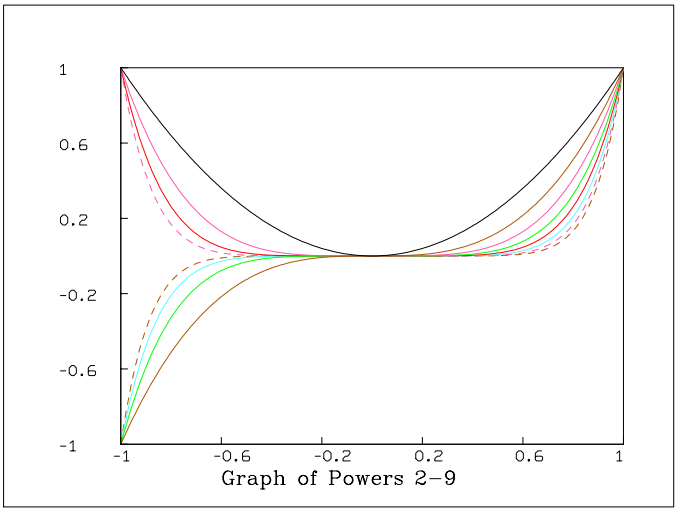

Figure B.44: Window `POTENZEN`.

B.20 RANDWALK.DO

The do-file RANDWALK.DO creates a vector of Gaussian random numbers. A random walk is created by doing a quadrature of a linear interpolation of these numbers.

```
/* FILE: RANDWALK.DO */
ECHODO = 3
RESET

/* RANDWALK.DO CREATES A VECTOR OF GAUSSIAN RANDOM
 NUMBERS. A RANDOM WALK IS CREATED BY DOING A
 QUADRATURE OF A LINEAR INTERPOLATION OF THESE
 NUMBERS.*/
M COL 1 = 1:100:1
START = NORMRAN(343,0,1)
FCT GO(X)=NORMRAN(0,0,1)
M COL 2 = GO ON M COL 1

DRAW M COLOR BROWN
BOTTOM TITLE "Gaussian Random Sequence" FONT 16
KARLF = W
VIEW KARLF

BLANK KARLF
/* DEFINE A LINEAR INTERPOLATION OF THE PINK NOISE.
 WHY IS THE NOISE CALLED PINK? */
FCT WALK'T(T) = LOOKUP(M,T)
INITIAL WALK(0)=0
N COL 1 = 1:100:0.25
NN = POINTS(WALK,N)
DRAW NN COLOR ORANGE
BOTTOM TITLE "Classic Random Walk" FONT 16
THEWALK = W
VIEW THEWALK

BLANK THEWALK
DRAW NN COLOR ORANGE
DRAW M COLOR BROWN
BOTTOM TITLE "Random Sequence & Walk" FONT 16
KOMBO=W
VIEW KOMBO
```

Window KARLF is shown in Figure B.45.
Window THEWALK is shown in Figure B.46.
Window KOMBO is shown in Figure B.47.

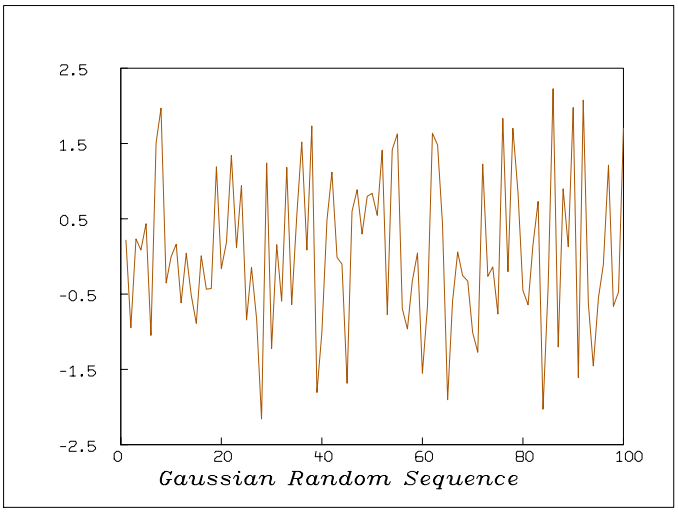

Figure B.45: Window KARLF.

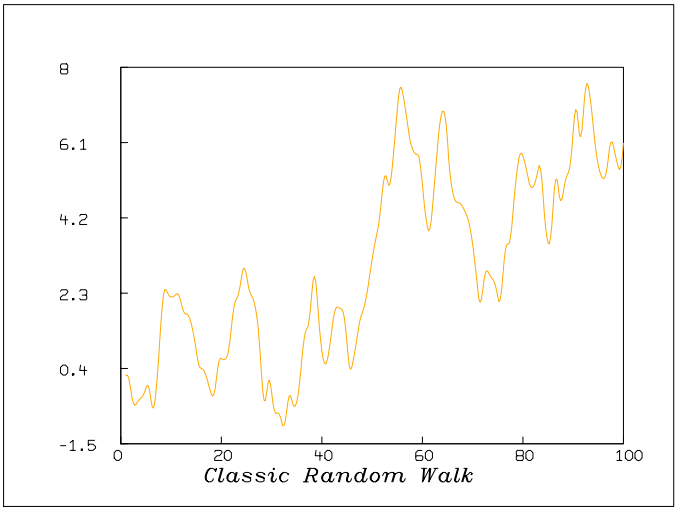

Figure B.46: Window THEWALK.

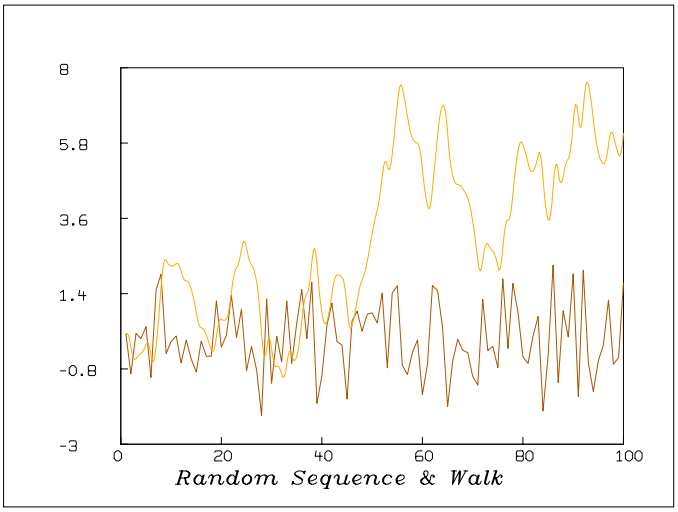

Figure B.47: Window KOMBO.

B.21 ROUNDOFF.DO

The do-file ROUNDOFF.DO compares nested and directly evaluated polynomials.

```
/* FILE: ROUNDOFF.DO */
ECHODO = 3
RESET

/* ROUNDOFF.DO COMPARES NESTED AND DIRECTLY EVALUATED
 POLYNOMIALS.*/

/* CREATE SOME POLYNOMIAL COEFFICIENTS:*/
C COL 1 = (5:1)

/* DEFINE DIRECT POLYNOMIAL EVALUATIONS:
 SUM UP OR DOWN, DEPENDING ON THE SIZE OF
 THE ARGUMENT.*/
FCT P(N,C,X)=IF ABS(X)>=1 THEN SUM(I,0,N,C(I+1)*X^I) \
 ELSE Q(N,C,X)
FCT Q(N,C,X)=IF ABS(X)>0 THEN SUM(I,0,N,C(N-I+1)*X^(N-I)) \
 ELSE C(1)

/* THE FOLLOWING SHOULD BE EQUAL.*/
PRAYER = " THESE SHOULD BE EQUAL.";
TYPE PRAYER
P(4,C,1)
1+2+3+4+5
PAUSE

/* DRAW A POLYNOMIAL:*/
NSTEPS = 301
X COL 1 = (-1:1!NSTEPS)
FCT POLY4(X) = P(4,C,X)
X COL 2 = POLY4 ON X COL 1
LBOX = 0.20
SQUEEZE = "YAXIS FORMAT(-3,4,0,0,4,0) OFFSET(-.17,-.01) IN W"
WEISS = "FRAME COLOR WHITE; XAXIS COLOR BLUE; YAXIS COLOR BLUE"
BLAU = "IMAGEBOX COLOR BLUE"
SCHWARZ = "BOTTOM TITLE COLOR BLACK FONT 21"
DRAW X COL (1,2) COLOR BROWN
DO WEISS; DO SCHWARZ; DO BLAU
BOTTOM TITLE "This is a polynomial"
```

```
P1 = W
VIEW P1

BLANK P1
/* DEFINE NESTED POLYNOMIALS RECURSIVELY:*/
FCT NR(N,M,C,X)=IF M>1 THEN X*NR(N,M-1,C,X)+C(N-M+1) \
 ELSE X*C(N+1)+C(N)
FCT NEST(N,C,X) = IF N <= 0 THEN C(1) ELSE NR(N,N,C,X)
TYPE PRAYER
NEST(4,C,1); P(4,C,1);
TYPE PRAYER;
NEST(4,C,0); P(4,C,0);
PAUSE

/* THIS FUNCTION SHOULD BE IDENTICALLY 0.*/
FCT DIFF4(XXX) = P(4,C,XXX) - NEST(4,C,XXX)
X COL 3 = DIFF4 ON X COL 1

/* DIFFERENCE NESTED AND DIRECT POLYNOMIALS
 ON [-1,+1].*/
LBOX = 0.24;
DRAW X COL (1,3) COLOR ORANGE
DO SQUEEZE; DO WEISS; DO SCHWARZ; DO BLAU
BOTTOM TITLE "Nested vs.  Direct Polynomial"
P2 = W
VIEW P2

BLANK P2
X10 COL 1 = 10*(X COL 1)
X10 COL 2 = DIFF4 ON X10 COL 1
DRAW X10 COL(1,2) COLOR ORANGE
DO SQUEEZE; DO WEISS; DO SCHWARZ; DO BLAU
BOTTOM TITLE "Polynomial Discrepancies on Wider Range"
/* DIFFERENCE NESTED AND DIRECT POLYNOMIALS
 ON [-10,+10].*/
P3 = W
VIEW P3
```

Window P1 is shown in Figure B.48.
Window P2 is shown in Figure B.49.
Window P3 is shown in Figure B.50.

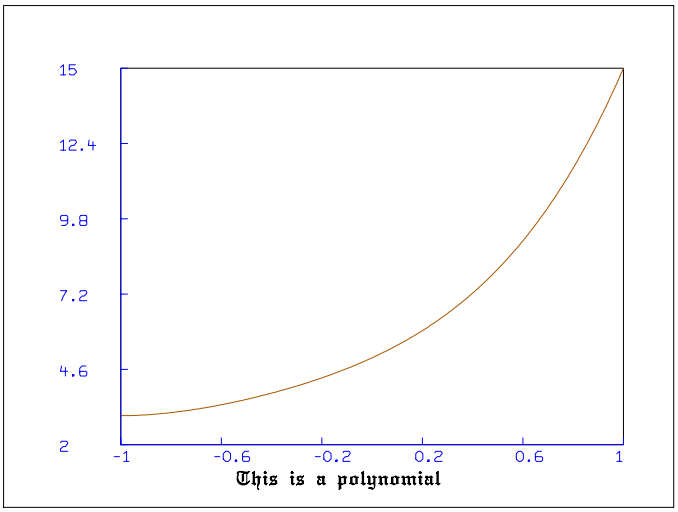

Figure B.48: Window P1.

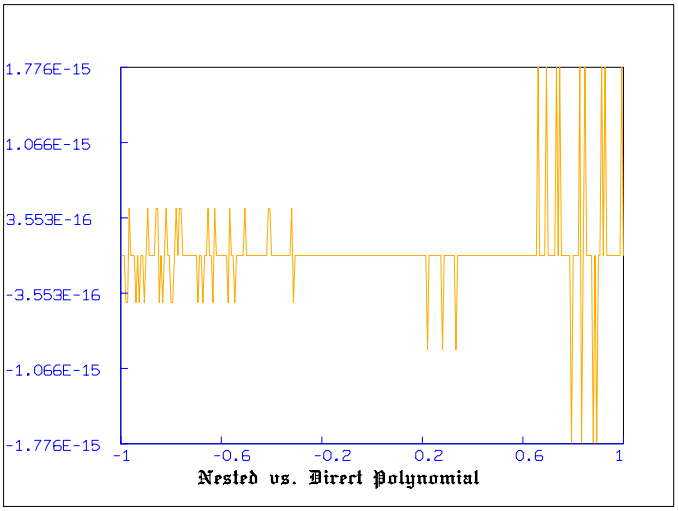

Figure B.49: Window P2.

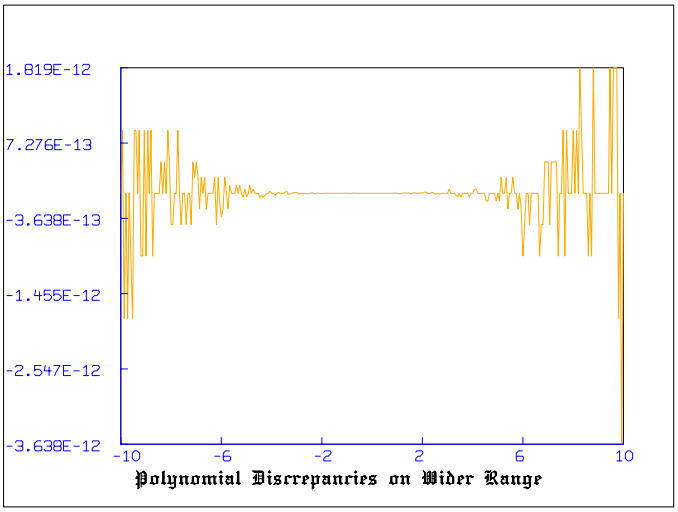

Figure B.50: Window P3.

B.22 SLABLF.DO

The do-file `SLABLF.DO` creates a graph of Legendre polynomials. The recurrence formula is applied to matrix columns, so this is more efficient than using the MLAB operator with a function, if you need the lower-order polynomials, too.

```
/* FILE: SLABLF.DO */
ECHODO = 3
RESET

/* THIS SLABLF.DO FILE CREATES A GRAPH OF
 LEGENDRE POLYNOMIALS. THE RECURRENCE
 FORMULA IS APPLIED TO MATRIX COLUMNS, SO
 THIS IS MORE EFFICIENT THAN USING THE MLAB
 ON OPERATOR WITH A FUNCTION, IF YOU NEED
 THE LOWER-ORDER POLYNOMIALS TOO.*/

/* SELECT A TITLE IN CELTIC STYLE:*/
TOP TITLE "Legendre Polynomials Degree 0 - 6" FONT 23
TOP TITLE AT (0, 0.97)

/* DEFINE THE LIMITS OF THE WINDOW WHICH WILL HOLD
 THE GRAPH:*/
WINDOW -1.  TO 1., -1.  TO 1.1

/* CREATE THE ARGUMENTS LIST FOR THE LEGENDRE
 POLYNOMIALS:*/
NSTEPS = 301
LEGPOL COL 1 ROW(1:NSTEPS) = 1
LEGPOL COL 2 = (-1:1!NSTEPS)

/* GENERATE LEGENDRE POLYNOMIALS DEGREES 0-6
 IN LEGPOL:*/
FCT N2(M) = (M+M-1)/M; FCT N1(M) = (M-1)/M
FOR I = 3:7 DO { CF2 = N2(I-1); CF1 = N1(I-1);
LEGPOL COL I = CF2*(LEGPOL COL 2)*'(LEGPOL COL (I-1));
LEGPOL COL I = LEGPOL COL I - CF1*LEGPOL COL (I-2); }

/* GRAPH THE LEGENDRE POLYNOMIALS IN DISTINCT
 COLORS:*/
DRAW LEGPOL COL(2,1) COLOR GREEN
DRAW LEGPOL COL(2,2) COLOR BROWN
```

```
DRAW LEGPOL COL(2,3) COLOR RED
DRAW LEGPOL COL(2,4) COLOR ORANGE
DRAW LEGPOL COL(2,5) COLOR VIOLET
DRAW LEGPOL COL(2,6) COLOR AQUA
DRAW LEGPOL COL(2,7) COLOR ROSE
SLABGRAF=W
VIEW SLABGRAF
```

Window `SLABGRAF` is shown in Figure B.51.

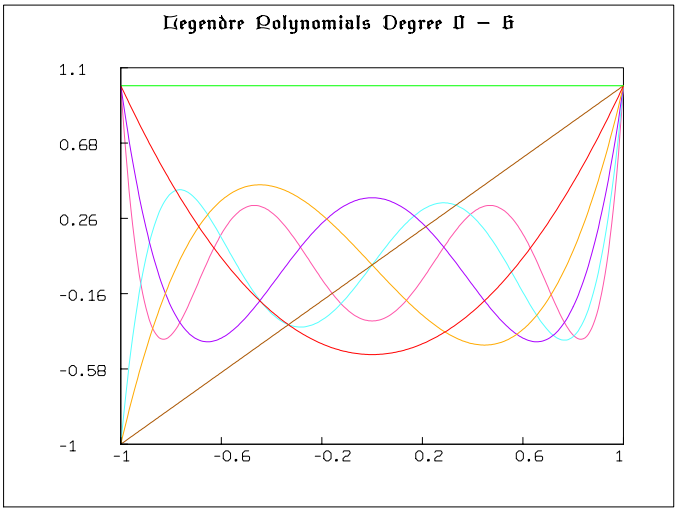

Figure B.51: Window SLABGRAF.

B.23 VINTI.DO

The do-file VINTI.DO demonstrates the need for a tight error tolerance for orbital mechanics. It first runs DIRKB.DO B.5 to test for orbit closure and then cuts the error control ERRFAC by a factor of 100. Plots are made of the energy and angular momentum, which should be constants.

The physicist John P. Vinti solved the artificial satellite problem in closed form.

```
/* FILE: VINTI.DO */
ECHODO = 3
RESET

/* ORBIT ANALYSIS PROGRAM VINTI.DO ILLUSTRATES
 THE NEED FOR A TIGHT ERROR TOLERANCE FOR ORBITAL
 MECHANICS. IT FIRST RUNS DIRKB.DO TO TEST FOR
 ORBIT CLOSURE AND THEN CUTS THE ERROR CONTROL
 ERRFAC BY A FACTOR OF 100.  PLOTS ARE MADE OF
 THE ENERGY AND ANGULAR MOMENTUM, WHICH SHOULD BE
 CONSTANTS.

 THE PHYSICIST JOHN P. VINTI SOLVED THE
 ARTIFICIAL SATELLITE PROBLEM IN CLOSED FORM.*/

/* INTEGRATION ERROR TOLERANCE IS: */
TYPE ERRFAC
PAUSE

DO DIRKB.DO
BLANK KEPLER
ERRFAC=ERRFAC/100

TYPE " INTEGRATION ERROR TOLERANCE NOW IS:"
TYPE ERRFAC
PAUSE
DO S
TOP TITLE "Precise Keplerian Motion" FONT 21, AT (0, .90)
DO ORBIT; DO SUN
JPVINTI=W
VIEW JPVINTI

BLANK JPVINTI
/* SELECT THE INITIAL CONDITIONS:*/
```

```
RR ROW (1:3) = M ROW 1 COL (2:6:2)
RDOT ROW (1:3) = M ROW 1 COL (3:7:2)

/* COMPUTE AND OUTPUT THE INITIAL ANGULAR MOMENTUM
 FIRST THE VECTOR VALUE:*/
CAPG = RR # RDOT

/* GET THE EUCLIDEAN NORM, AKA THE ABSOLUTE VALUE:*/
GEE = MNORM(CAPG,0); TYPE GEE

/* SORT OUT POSITIONS AND VELOCITIES:*/
RRR = M COL (2,4,6);
RRRDOT = M COL (3,5,7);
GG1 = LENGTH(RRR # RRRDOT)
PAUSE

/* REWORK MATRIX GG1 TO PUT TIME IN COLUMN 1
 FOR GRAPHS:*/
GG1 COL 2 = GG1 COL 1
GG1 COL 1 = M COL 1
DRAW GG1 COL (1,2) COLOR BROWN
GG2(1,1) = M(1,1); GG2(2,1) = M(NROWS(M),1)
GG2(1,2) = GG1(1,2); GG2(2,2) = GG2(1,2)
DRAW GG2 COL (1,2) LT DASHED COLOR ORANGE

YAXIS FORMAT(-3,7,0,0,4,0) OFFSET(-0.15,-0.01) IN W
TOP TITLE "Angular Momentum" FONT 21
GGRAPH = W
VIEW GGRAPH

BLANK GGRAPH
/* DISPLAY THE HAMILTONIAN (ENERGY) FUNCTION:*/
TYPE H

/* CONCATENATE VELOCITIES TO POSITIONS TO EVALUATE H():*/
RRR = RRR & 'RRRDOT
ENERGY = H ON RRR
/* ADD THE TIME REFERENCE */
ENERGY COL 2 = ENERGY COL 1
ENERGY COL 1 = M COL 1
GG2(1,2) = ENERGY(1,2); GG2(2,2) = GG2(1,2)
DRAW ENERGY COL(1,2) COLOR BROWN
```

```
YAXIS FORMAT(-3,7,0,0,4,0) OFFSET(-0.15,-0.01) IN W
DRAW GG2 COL (1,2) LT DASHED COLOR ORANGE
TOP TITLE "Energy Integral 2-Body Motion" FONT 21
VISVIVAE=W
VIEW VISVIVAE

TYPE " YOUR GRAPHICS ARE HERE:"; TYPE WINDOWS;
```

Window `JPVINTI` is shown in Figure B.52.
Window `GGRAPH` is shown in Figure B.53.
Window `VISVIVAE` is shown in Figure B.54.

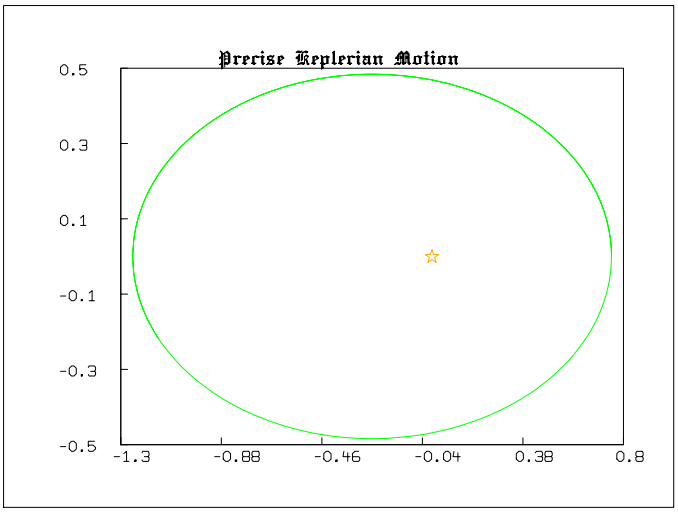

Figure B.52: Window `JPVINTI`.

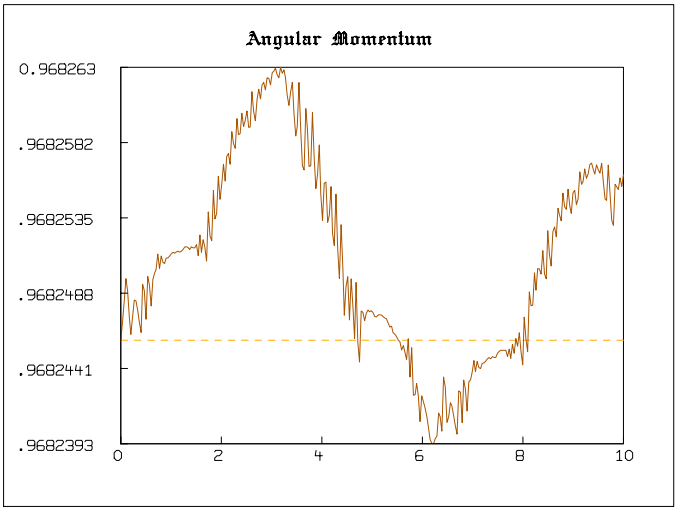

Figure B.53: Window `GGRAPH`.

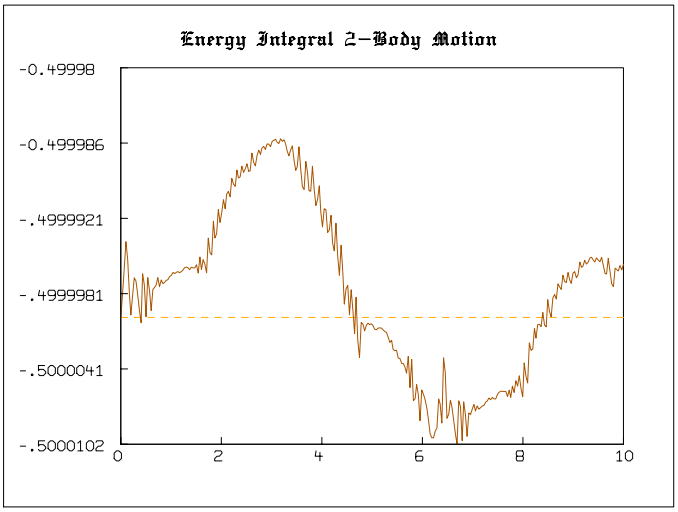

Figure B.54: Window `VISVIVAE`.

B.24 VOLTERRA.DO

The do-file `VOLTERRA.DO` illustrates the mathematical properties of the classic Volterra predator-prey equations. They have an integral, here `FCT CAPK()`, which is analogous to energy in mechanics. Try to recast these equations in Hamiltonian form. (Hint: Use log X and log Y as variables.)

```
/* FILE: VOLTERRA.DO */
ECHODO = 3
RESET

/* PREDATOR(Y)-PREY(X) MODEL
 THIS DEMO ILLUSTRATES THE MATHEMATICAL PROPERTIES
 OF THE CLASSIC VOLTERRA PREDATOR-PREY EQUATIONS.
 THEY HAVE AN INTEGRAL, HERE FCT CAPK(), WHICH IS
 ANALOGOUS TO ENERGY IN MECHANICS. TRY TO RECAST
 THESE EQUATION IN HAMILTONIAN FORM. (HINT: USE
 LN X AND LN Y AS VARIABLES.) */
FCT Y 'T(T)=K*X*Y-C*Y
FCT X 'T(T)=A*X-B*X*Y
INIT X(0)=X0
INIT Y(0)=Y0
Y0 = 1; X0 = .8
K = 1; C = 1; A = 1; B = 1
S = "SVECTR=INTEGRATE(X,Y,T0:TF!NSTEPS)"
T0 = 0; TF = 8; NSTEPS = 576

/* INTEGRATE THESE COUPLED, NONLINEAR ODES:...*/
DO S
FCT CAPK(X,Y)=B*X - A*LN(B*X/A) + K*Y - C*LN(K*Y/C)

/* PLOT A TIME SERIES GRAPH OF BOTH VARIABLES:...*/
TOP TITLE "Predator-Prey Time Series" FONT 11
DRAW SVECTR COL (1,2) COLOR RED
DRAW SVECTR COL (1,4) COLOR GREEN
PPTSER = W
VIEW PPTSER

BLANK PPTSER
/* EVALUATE THE INTEGRAL ON THE NUMERICAL SOLUTION:...*/
SVXY COL 1 = SVECTR COL 2
SVXY COL 2 = SVECTR COL 4
```

```
KAPP = CAPK ON SVXY
SVXY COL 2 = KAPP
SVXY COL 1 = SVECTR COL 1
DEL KAPP
REFF COL 1 ROW 1 = T0
REFF COL 1 ROW 2 = TF
REFF COL 2 ROW 1 = CAPK(X0,Y0)
REFF COL 2 ROW 2 = REFF(1,2)
DRAW REFF LT LDASH COLOR RED

/* FIDGET WITH THE GRAPH:...*/
YAXIS FORMAT(-3,7,0,0,4,0) OFFSET(-0.15,-0.01) IN W
DRAW SVXY COL(1,2) COLOR BROWN
TOP TITLE "Volterra Energy Analog" FONT 11
PPINT = W
VIEW PPINT

BLANK PPINT
/* PLOT THE PHASE PLANE DIAGRAM ...
 WHICH IS A CONTOUR OF THE INTEGRAL:...*/
DRAW SVECTR COL (2,4) COLOR BROWN
TOP TITLE "Volterra Phase Plane" FONT 11
PPHASE = W
VIEW PPHASE

TYPE "SAVE THE GRAPHS (WINDOWS), IF YOU WISH."
TYPE WINDOWS
```

Window PPTSER is shown in Figure B.55.
Window PPINT is shown in Figure B.56.
Window PPHASE is shown in Figure B.57.

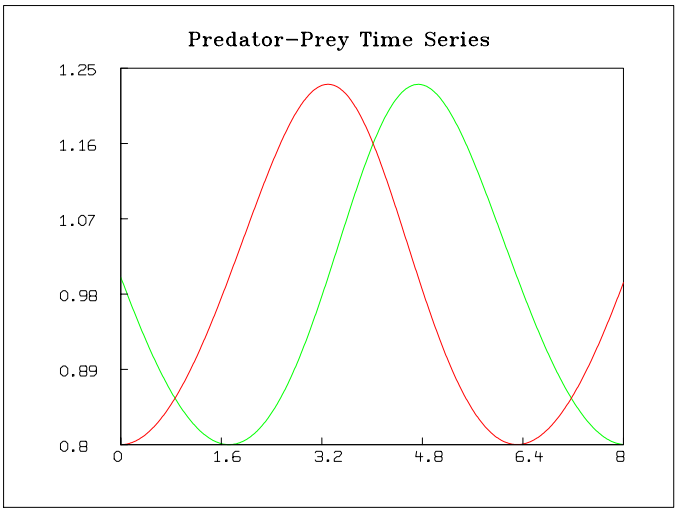

Figure B.55: Window PPTSER.

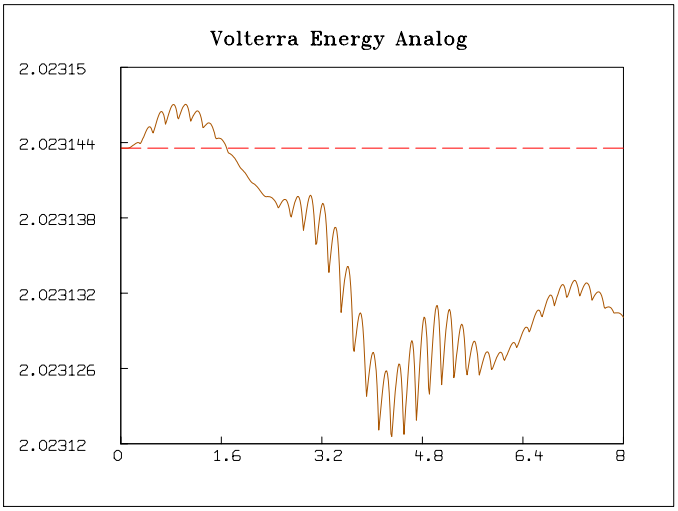

Figure B.56: Window PPINT.

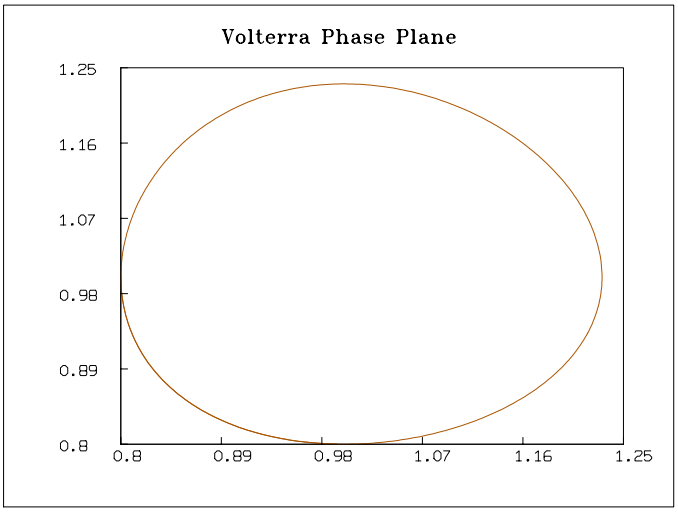

Figure B.57: Window `PPHASE`.

List of Figures

A list of figures generated by the DO-files in Appendix B begins on the next page. Positioning the mouse cursor on an entry in the list and clicking the mouse button will display the corresponding figure.

List of Figures

Appendix C

Index

The index begins on the next page. Index entries in `BOLD` type are MLAB function names or commands. Positioning the mouse cursor on a page number–appearing blue in color and following an index entry–and clicking the mouse button will display the corresponding page.