Invitation to Dynamical Systems
Software Guide

This document describes the software for use with the text *Invitation to Dynamical Systems* by Edward Scheinerman published by Prentice-Hall.

1 Overview

The software described in this guide is designed to be used with *Invitation to Dynamical Systems*, an undergraduate textbook on dynamical systems, chaos, and fractals.

These programs are designed to be used with MATLAB (version 4 or later) including the student version of MATLAB. MATLAB is available for Macintosh, Windows, and X-windows. The files are plain text files which can be read and modified by the user, and run unmodified on all three platforms.

The software contains the following packages of programs:

- `sys`—a collection of programs to simulate dynamical systems and present the results as graphs and/or phase diagrams.
- `gradsys`—a collection of programs to simulate continuous time dynamical systems which arise from gradient systems.
- `web`—a program to create "web" diagrams for one dimensional discrete time systems.
- `bif`—a collection of programs to create bifurcation diagrams.
- `complex`—a collection of programs to illustrate the ideas in Chapter 6, Complex Dynamical Systems.
- `fractory`—a fractal drawing environment.
- `misc`—various other useful programs.

The programs for these sections are contained in their own subdirectories (folders). To use one of the packages, enter the directory containing the programs you wish to use and then start MATLAB. (Alternatively, start MATLAB first, and use MATLAB's `cd` command to enter the appropriate directory.)

2 sys

The programs in the directory `sys` are designed to illustrate both continuous and discrete time dynamical systems, i.e., systems of the form

$$\mathbf{x}' = f(\mathbf{x}) \qquad \text{or} \qquad \mathbf{x}(k+1) = f(\mathbf{x}(k)).$$

The two main commands you use are `cont` for continuous time systems and `discr` for discrete time systems. To simulate a dynamical system you need to (1) write a short program to calculate the function f, (2) use either the `cont` or `discr` commands to compute the trajectory of the system, and then (3) view the results. Let's go through these steps one by one.

Suppose we want to study the dynamics of a simple pendulum. The system is of the form $\mathbf{x}' = f(\mathbf{x})$ where

$$f\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin x_1 \end{bmatrix}.$$

Our first step is to prepare a MATLAB "`.m`" file which calculates the function f. Such a file is in the `sys` directory and is called `pendulum.m`. (Note that MATLAB files have the `.m` suffix.) Here's a copy of that file:

```
function xout = pendulum(x)
%
% pendulum example
%
xout(1) = x(2);
xout(2) = -sin(x(1));
```

The second step in the process is to use the `cont` command to compute the trajectory of the system. Type the command `cont` and the computer responds

```
Enter name of the function (rhs)->
```

Please type `pendulum` and hit return. Next the computer asks

```
Please enter the initial vector ->
```

Let's say we want to start at $\mathbf{x}_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$. We input this by typing `[2 0]` and pressing return. Next the computer asks for the starting time:

```
Please enter the starting time (default 0) ->
```

You may either enter `0` or just press return. Finally, it asks for the ending time:

```
Please enter the ending time (default 1) ->
```

Enter a value (say `15`) and press return again.

After a wait, the computer plots graphs of the functions $x_1(t)$ and $x_2(t)$ for the range $0 \le t \le 15$.

You may now view your results in phase space. To do this, give the command `phase2c` and the computer draws the trajectory of the system atop a vectorfield.

The "2" in `phase2c` stands for 2-dimensional and the "c" stands for continuous. The command `phase3c` draws the trajectory for 3-dimensional, continuous time systems. (Try the `lorenz` system, for example.)

The command discr is quite similar to cont, but is used to emulate discrete time dynamical systems. A nice example of such a system is $\mathbf{x}(k+1) = f(\mathbf{x}(k))$ where

$$f\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -0.64 & -0.48 \\ 0.48 & -0.64 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The steps for using discr are quite similar to those for using cont. First, prepare a file which computes f. This particular example is in the file lin2f.m which we repeat here:

```
function y = lin2f(x)
%
% eigenvalues of A are   -0.6400 +/- 0.4800i
% which have absolute value 0.8
%

A = [
   -0.6400   -0.4800
    0.4800   -0.6400
];

b = [1;1];

y = A*x + b;
```

Next, give the discr command. Follow the prompts typing lin2f for the name of the function, say [2 3] for the starting vector, 0 for the starting time and 20 for the ending time. The computer then plots graphs of $x_1(k)$ and $x_2(k)$ for $k = 0, 1, 2, \ldots, 20$.

The phase diagram for this system can now be plotted with the command phase2d. Use phase3d for 3-dimensional, discrete time systems.

There are a variety of example functions included in the sys directory which you are encouraged to explore.

In addition, the command vf is used to draw vector fields of functions. Let $f\colon \mathbf{R}^2 \to \mathbf{R}^2$. We use vf to plot, for each point $\mathbf{x}$ in the plane, an arrow anchored at $\mathbf{x}$ and pointing in the direction $f(\mathbf{x})$.

For example, let $f(\mathbf{x})$ be given by

$$f\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 - x_1^3 \\ -x_1 \end{bmatrix}.$$

The file example1.m is available for computing this function. Now give the command vf. When asked for name of the function, reply example1. Next you are asked for the range of the x and y values (these are the ranges for x_1 and x_2 respectively). Enter [-1 1] for both. The computer now draws a picture of the vectorfield f in this range.

3 gradsys

The collection of programs in the directory gradsys is used to emulate gradient systems (see §3.3.4). Let $f\colon \mathbf{R}^2 \to \mathbf{R}$. A gradient system is one of the form

$$\mathbf{x}' = -\nabla f(\mathbf{x}) \quad \text{where} \quad \nabla f = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{bmatrix}.$$

The command gradsys is used to compute trajectories and plot graphs of gradient systems. You need to supply to MATLAB two .m files: one to compute f and another to compute ∇f.

For example, suppose f is given by

$$f(\mathbf{x}) = f\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \sin x_1 \sin x_2 + 1.1$$

so the gradient ∇f is given by

$$\nabla f(\mathbf{x}) = \nabla f\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \cos x_1 \sin x_2 \\ \sin x_1 \cos x_2 \end{bmatrix}.$$

The graph of $z = f(x, y)$ looks something like an egg crate.

We write files fegg.m to compute f and gegg.m to compute f's gradient, ∇f. Here is the file fegg.m (which is included in the gradsys directory):

```
function y = fegg(x)
%
% egg crate surface
%
y = sin(x(1)) * sin(x(2)) + 1.1 ;
```

and here is the file gegg.m:

```
function y = gegg(x)
%
% gradient for the egg crate surface
%
y(1) = cos(x(1))*sin(x(2));
y(2) = sin(x(1))*cos(x(2));
```

Once these files are created, we are ready to give the gradsys command. Do so now. The computer asks you for the name of the function (enter fegg) and the name of the function's gradient (enter gegg). Then it asks for a starting vector (enter [1.2 -.3], or whatever you like), and then for start and stop times for time (enter, say, 0 and 2).

The computer now works a bit and then plots a fancy 3-dimensional graph. On the horizontal xy-plane we see the vector field defined by $-\nabla f$, contour lines corresponding to curves on which f is constant, and the trajectory of the system. In three dimensions, we see the graph of the function f, and, projected onto this surface, the trajectory of the system.

4 `web`

Let $f: \mathbf{R} \to \mathbf{R}$ and consider the discrete time system, $x(k+1) = f(x(k))$. The successive values $x(0)$, $x(1)$, $x(2)$, etc., are the iterations of f, i.e. x_0, $f(x_0)$, $f^2(x_0)$, etc. These values can be found graphically as in Figures 2.1–2.6 and 3.4–3.5. The `web` program in the `web` directory (folder) can be used to plot such diagrams.

To use `web` you need to prepare a MATLAB `.m` file to compute f. For example, if $f(x) = 2.9x(1-x)$ then create a file called `rhs.m` (or another name you choose) that looks like this:

```
function y = rhs(x)
%
y = 2.9 * x .* (1-x);
```

(Note: this exact file already exists in the `web` directory as an example and is, in fact, called `rhs.m`.)

Now give the command `web`. The computer asks:

```
Enter name of the function (rhs)->
```

Type `rhs` and press return. Now the computer asks:

```
Please enter the initial value ->
```

Enter `0.5` and hit return. Finally, the computer asks you to enter the number of iterations. Enter a number (or just press return for 10 iterations). After a brief wait, the computer plots the web diagram.

Important note: To plot the graph of the function f, for efficiency, MATLAB computes f for an entire *list* of values (not just one value at a time). Thus, in the program for f above we wrote:

```
y = 2.9 * x .* (1-x);
```

Notice that we used `.*` to mutliply `x` and `1-x`. This is because MATLAB uses `x` as a list of values (a vector). Had we used `*` in place of `.*`, MATLAB would have thought we meant matrix multiplication.

5 `bif`

The collection of programs in the directory `bif` is used to create bifurcation diagrams. Let $f_a: \mathbf{R} \to \mathbf{R}$ denote a family of functions; the subscript a stands for a parameter which gradually changes the function f_a as a is changed. (For a discussion of bifurcation diagrams, see Chapter 4.)

The first step is to prepare a file to compute the functions f_a. Note that to compute $f_a(x)$ there are *two* inputs: the argument x and the parameter a. For example, for the quadratic family of functions $f_a(x) = x^2 + a$, we prepare the following `.m` file:

```
function y = quad(x,a)
%
y = x .* x + a;
```

Two comments about this file: First, the function `quad` takes two inputs, the first, `x`, is the argument and the second, `a`, is the parameter we vary. Second, notice that when we multiply `x` with itself (to square it) we use `x.*x` and not `x*x` (the difference is the dot in front of the asterisk). This is a MATLAB technicality. The `.*` multiplication allows `x` to be a list of numbers (not just a single number). The ability of `quad` to work on entire lists of numbers greatly speeds up the calculations.

Now that the file `quad.m` is ready, give the `bif` command. The computer prompts you for the name of the function (enter `quad`), the range of the parameter value (say, from `-1.5` to `0.2`), and the range of x values you expect to see (say, from `-2` to `2`). After a delay, the computer displays the bifurcation diagram on the screen.

In the file `bif.m`, you can find the following lines near the top:

```
pres = 90;
xres = 90;
```

These values control the resolution of the picture. As their names suggest, `pres` controls the resolution of the parameter range, and `xres` controls the resolution of the x-range. Larger values give better pictures, but need more time and memory for the computations.

The file `slowbif.m` is also included in this directory. It performs the same computations as `bif.m`, but is less efficient. Its slight advantage is that it performs computations single numbers at a time and so users can write function files which do not need to use `.*` in place of `*`, etc.

6 `complex`

The programs in the directory `complex` are used to illustrate concepts in complex dynamical systems (see Chapter 6).

- `base`: The program `base` computes and plots the fractal defined as follows. Let b be a complex number and let $C(b)$ denote the set of all numbers of the form $0.d_1d_2d_3\cdots{}_b$ where each d_i is 0 or 1 (see §6.4). One should have $|b| > 1$ to assure convergence.

  When you type `base`, the computer asks for the complex number b (reply, say `1.2 - .4i`) and the number of digits (say, `10`). `base` creates an array of size 2^d (where d is the number of digits), so if $d > 12$ the array will be too big for the student edition of MATLAB.

If the number of digits is small, the computations will be fast, but the picture will not be very pleasing. On the other hand, a large value of d results in the computer needing a lot of memory to do its computations, but gives better pictures.

Real numbers b result in less interesting pictures, but try $b = 3$ to get a picture of the Cantor set.

- `julia`: The program `julia` is used to compute pictures of Julia sets (see §6.1). Let $f_c(z) = z^2 + c$ and $B_c = \{z : |f_c^k(z)| \not\to \infty \text{ as } k \to \infty\}$. After giving the `julia` command, the computer asks you for the parameter c (enter, say `-.7+.25i`), the lower left corner of the image (enter, say `-1.5-i`), the upper right corner of the image (enter, say `1.5+i`), the escape time (enter, say `30`), and the horizontal resolution (enter, say `50`). After a wait, an image of the Julia set B_c appears on the screen.

  A larger escape time value gives a more accurate picture, and a larger resolution value gives sharper image. Larger values, however, require more memory and more time.

  At the end of the file `julia.m` are commands to set MATLAB's colormap; one is better for color monitors and the other is better for grayscale. Edit the file accordingly.

  The command `julia` uses some MATLAB tricks to speed up execution. A more understandable version of this program, `jslow`, is also included in the `complex` directory.

  Although `julia` is faster than `jslow`, it is not very fast. These programs are not recommended for high resolution images.

- `mandel`: Similar to `julia`, this program is used to create images of the Mandelbrot set $\mathcal{M}$ (see §6.2). After giving the command `mandel`, the computer asks for the lower left corner of the image (enter, say `-2-1.5i`), the upper right corner (enter, say `.5+1.5i`), the escape time (enter, say `30`), and the horizontal resolution (enter, say `50`). After a wait, the computer draws a picture of the Mandelbrot set.

  At the end of the file `mandel.m` are commands to set MATLAB's colormap; one is better for color monitors and the other is better for grayscale. Edit the file accordingly.

  The `mandel` command uses some MATLAB tricks. Another version of this program, `mslow`, also draws the Mandelbrot set, but is easier to understand.

  Although `mandel` is faster than `mslow`, it is not very fast. These programs are not recommended for high resolution images.

- `newton`: The command `newton` computes the complex Newton picture (see §6.3).

  When you run `newton`, the computer asks for a resolution. Smaller numbers result in faster computations using less memory, but the picture is rather coarse. Larger values result in longer computations requiring more memory, but produce nicer pictures. Use an odd number for the resolution as this prevents 0 from being a starting value in the Newton iteration.

  At the end of the file `newton.m` are commands to set MATLAB's colormap; one is better for color monitors and the other is better for grayscale. Edit the file accordingly.

  If you want to produce a high resolution picture, use the `newton2` command instead. It uses memory somewhat more efficiently than `newton`.

  Both `newton` and `newton2` take advantage of the fact that MATLAB works most efficiently doing matrix-at-a-time computations (as opposed to individual entry computations). This makes the files `newton.m` and `newton2.m` hard to understand. A third version, `newtslow.m` does the same computations but it is easier to see what is going on in the program. However, it is very much slower than the other versions and not recommended even for moderate resolutions.

7 fractory

`fractory` is an elaborate tool for creating fractals from iterated function systems (IFS's) of affine functions (see Chapter 5).

The user works in a graphical environment, manipulating buttons, dials, and switches with the mouse. Through this graphical user interface the user enters transformations of the form

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}.$$

Such transformations can be described in a variety of ways. One way is to directly enter the six numbers a through f; indeed, fractory allows the user to enter the numbers in just this way (press the `Ax+b` button), but there are a variety of other methods for setting these functions.

Think of the affine functions as geometric transformations. As such, we can apply the function to the unit square (whose lower left corner is at the origin and whose upper right corner is at the point $(1,1)$). An affine function transforms the unit square into a parallelogram. With the mouse, the user can "grab" the corners of the unit square and drag them to any position. By setting the location of the corners, the affine function is determined. Alternatively, the user can use the various controls in fractory to define affine transformations by horizontal and vertical rescalings, rotations, skewing, and translations.

After the first function has been set, more functions can be added. When all the functions in the IFS have been set, a simple button press commands `fractory` to draw the fractal.

The best way to learn how to use `fractory` is to try it and enjoy. The `save` and `load` buttons enable the user to transfer IFS data to and from the disk. A variety of example IFS's are saved on disk; try loading them and computing their fractals.

The `help` button in `fractory` can provide you with much more information.

The `fractory` system makes a fairly high demand on the computer's resources. Its use with slower machines might be frustrating. A simpler fractal drawing program, called `fraclite`, is also included in the `fractory` directory. To use `fraclite`, it is convenient to define a matrix to hold the IFS fractal data. For example, for Sierpinski's triangle the data is a 3×6 matrix which you can create by typing:

```
sier = [.5 0 0 .5 0 0
 .5 0 0 .5 .5 0
 .5 0 0 .5 .25 .5]
```

Now that you have saved the data in the variable `sier`, give the command `fraclite`. The computer first asks for the IFS data matrix (enter `sier`, or type the matrix in directly). Next the computer asks for how many points to plot (enter, say 2000). After a wait, the computer plots an image of the fractal using the random method described in §5.5.3.

8 misc

This directory contains a variety of other useful programs which we describe briefly here.

- `dance`: This command produces the "triangle dance" from §5.5.2. Find the line `npoints = 500;` and increase the number above 500 for a more detailed (but slower) picture.

- `markov`: This command is used to emulate Markov chains (see §2.3). To use the command, prepare a matrix `P` containing the transition matrix of the Markov chain. Make sure the rows of your matrix sum to one. For example, you may type

  ```
  P = [ .1 .1 .8
         0 .5 .5
        .6 .3 .1 ]
  ```

  Now give the command `markov`. It asks you to enter the transition matrix (answer `P` or type in a matrix) and then the number of *hundreds* of iterations to perform (e.g., if you want to run 2000 iterations, enter 20).

  The program now starts the Markov chain in state 1 and randomly moves from state to state depending on the transition matrix you entered. Every 100 iteration the compute presents a bar graph showing the frequencies with which the Markov chain visited each state.

  When the program is finished running, the variable `counts` holds the number of times the chain visited each state.

- `rk`: The `misc` directory contains the files `rk.c` and `rk.m`; both for performing Runge-Kutta numerical solution to differential equations (see §B.1.2).

  The `rk.c` file is a C language program (*not* a MATLAB program). The `rk.m` file is a MATLAB version. Both are designed to solve the differential equation $y' = t - y^2$ with $y(0) = 1$ in the range $0 \le t \le 2$. The file `rkf.m` computes the righthand side, $t - y^2$, and is used by `rk.m`.

  To use `rk.m` you need to run MATLAB (give the command `rk` to MATLAB). You can edit the file `rk.m` to change the step size, the range of t, or the initial conditions. You can edit the file `rkf.m` to change the righthand side of the differential equation.

  The `rk` command is meant to illustrate how the Runge-Kutta method works. For greater speed, try MATLAB's built-in commands `ode23` or `ode45`. Try typing the following:

  ```
  [t,y] = ode45('rkf', 0, 2, 1);
  plot(t,y)
  ```

  The file `rk.c` is a C language version of the Runge-Kutta method. To use it, your computer needs to have a C compiler. This version will be *much* faster than the MATLAB version. (However, for this small example, the difference is unimportant.)

- `shuffle`: Use this program to perform the GSR/shift shuffle described in §4.3. `shuffle` is a MATLAB function which you invoke by typing `shuffle(`*list*` )` where *list* is a list of numbers. The program then performs a GSR/shift shuffle on your list. For example try: `shuffle(1:10)` to shuffle a "deck of numbers" from 1 to 10.

  The `shuffle` function takes an optional second argument which gives the number of times to repeat the shuffle, for example `shuffle(1:52,20)` shuffles the list of numbers 1 through 52 twenty times.

9 Updates

While every attempt has been made to make sure that the programs run properly on any platform, and with the standard and student versions of MATLAB (4.0 or higher), it is possible that some incompatibilities might exist. If so, please inform the author (addresses below) of these problems and every attempt will be made to effect a remedy.

As revisions are made, the new versions will be made over the Internet via FTP from the MathWorks. Connect your FTP client to `ftp.mathworks.com` and go to the `/pub/books/scheinerman` directory. Alternatively, use FTP to connect to `brutus.mts.jhu.edu` and go to the `/pub/scheinerman/invite/software` directory.

The software can also be accessed on the Internet via the World Wide Web. You can visit the *Invitation to Dynamical Systems* home page at `http://www.mts.jhu.edu/~ers/invite.html`.

Instructors who use *Invitation to Dynamical Systems* in their course may download this software and distribute it to their students.

10 Copyright

 This software is free to be used by anyone who owns a copy of *Invitation to Dynamical Systems* by Edward Scheinerman.

Professor Edward R. Scheinerman
Department of Mathematical Sciences
The Johns Hopkins University
Baltimore, Maryland 21218 USA
Fax: (410) 516-7459
E-mail: `ers@jhu.edu`
Web: `http://www.mts.jhu.edu/~ers/ers.html`