GRIND Tutorial

Rob J. de Boer,
Theoretical Biology,
Utrecht University,
Email: R.J.DeBoer@bio.uu.nl

This document accompanies GRIND96.6

1 Introduction

This tutorial introduces you to GRIND and its mostly used commands. GRIND is a command line driven system for analyzing models in terms of differential or difference equations. GRIND consists of two parts. The first part is a preprocessor reading your model. The second part is the command line driven system in which you set (or vary) parameters and initial conditions, and in which you perform the numerical analysis of your choice. GRIND commands have long names but can be abbreviated to the first two (or more) characters of the name of the command.

2 Model definition

Use any text editor (e.g., emacs, xedit, vi) to define your model by writing a plain ASCII file of the following format

```
fc = c^2/(h^2 + c^2);
c '= r*c*(1-c) - b*n*fc;
n '= b*n*fc - d*n;
```

where `c` and `n` are two variables (i.e., the prey and the predator of an ecological model), and `fc` is an algebraic expression (for the sigmoid functional response). Algebraic expressions have to be defined before the differential equations. The GRIND preprocessor will also understand you if you use functions like `sin()`, `cos()`, `log()`, `exp()`, `sqrt()`, `abs()`, `mod(x,y)`, `max(x,y)` and `min(x,y)`.

3 Model analysis

When GRIND is installed properly you start GRIND by typing

```
 grind model
```

where we assume that the file that you made above is called `model`. GRIND will check if your model has been pre-processed before, and will start up its command line driven interface. Just wait until GRIND prompts you by something like

```
 G R I N D model analysis:
```

after this you just start typing commands like

```
r=1
b=.25
h=.1
d=.1
c=0.1
n=0.1
par
```

which initializes all parameters, the initial condition, and gives an overview of all parameter values. Parameter values can be stored for later usage by a command like

```
 par parfile
```

and can be read by GRIND by a command like

```
 read parfile
```

In fact you can read files with all sorts of GRIND commands. By the way you can quit from GRIND by the `bye` command.

4 Numerical integration

Now that the parameters and the initial condition are defined one can run the model by a numerical integrator

```
finish 20 20
run
```

will "run" your model for 20 time steps printing some output in 20 intervals (i.e., at every time step). The variables that are printed can be modified with the `output` command. GRIND has several graphics interfaces. We here assume that you are using some X11 interface. Then we can plot the output of this numerical integration by

```
terminal x11
axis v 0 2
timeplot
```

which tells GRIND that you are on an X11 terminal, which scales the vertical axis of the time plot between zero and two, and which plots the data of the previous run. First do a `run` then call `timeplot`!

5 Phase space analysis

The `axis` command can also be used for defining a two-dimensional (or three-dimensional) state space.

```
axis x c 1e-3 1
axis y n 1e-3 2
```

which defines a state space by two axis commands. The horizontal "x" axis is defined by the prey `c`, and runs from `c=0.001` to `c=1`. The vertical "y" axis is defined by the predator `n` and runs from `n=0.001` to `n=2`. For three-dimensional state spaces it is often useful to avoid an axis origin of zero because often the faces of the state space will be filled in as being a nullcline. The following commands will draw a two-dimensional frame on your screen, then plots two nullclines, and finally a vector field.

```
2d
nullcline c n
vector
```

Probably you will find the nullclines too sketchy. Then increase the accuracy of the nullclines, erase, and draw everything again by

```
nullcline 3
e2n
```

where the latter is a predefined macro for `erase;2d;nullcline`. Since this now looks nice you can plot a trajectory in this two-dimensional space by typing

```
finish 25 50
run
```

which gives you a simulation of 25 time steps plotted at 50 intervals. If you would like to continue this run just type

```
keep
run
```

where the `keep` command is used for copying the last state into the initial condition. Now we can change a parameter and start all over again:

```
h=.25
e2n
run
```

6 Equilibrium analysis

From any initial condition you can jump to an equilibrium point (stable or unstable!) by calling the Newton-Raphson algorithm. Here we first pick an initial condition close to the non-trivial equilibrium point defined by the intersection of the nullclines by activating the X11 cursor

```
cursor
newton
eigen
```

which asks you to click in the graphics window, finds an equilibrium starting from the point where you clicked, and reports the eigen values of the Jacobian matrix of the equilibrium point.

7 Three-dimensional

Defining a third "Z" axis, and calling the `3d` command everything becomes three-dimensional. To illustrate this, and to illustrate that an axis can also be defined by a parameter, you could type

```
axis z h 1e-3 1
e3n
shade c
```

8 Equilibrium continuation

For both maps and differential equations, GRIND allows for a very primitive continuation of equilibria. Having found an equilibrium by the `NEWTON` command, you can define one of the axes as a parameter, and continue that equilibrium along that axis. For instance

```
newton
axis x h 0 3
axis y n 0 5
erase
2d
continue x
```

which plots the equilibrium value of `n` for various values of h.

9 Input & Output

We have seen that files with GRIND commands can be written and be read. One can save the data from a numerical integration by supplying a file name to the run command, e.g., `run data1`. Additionally, all the graphics output can be saved in PostScript files by opening a second terminal, e.g., `terminal postscript figures.ps`, where `figures.ps` is the filename of the file with the PostScript output. The Postscript terminal can be closed by `terminal postscript off`.

10 White Noise

You can set noise on a parameter by typing

```
noise r 1 0.1
```

which gives the parameter r an average of one and a standard deviation of 0.1. Unfortunately noise can be set on one parameter only. Random drawings are done at every output step of the integrator! Thus, when the noise is on `finish 100 50` will give different results from `finish 100 100`. Switch the noise off by `noise off`.

11 Help

GRIND has a rudimentary help facility which you activate by typing

```
help command
```

where `command` is the name of the command you need some information about. For detailed information you will have to consult the GRIND manual.

12 Maps or Difference equations

The model definition of a map follows that of differential equations. Thus by writing `x '= a;` we mean that the change $\Delta x = a$, where $\Delta x = x_{t+1} - x_t$. For example, the standard logistic map $x_{t+1} = rx_t(1 - x_t)$ has to be written as

```
x '= r*x*(1 - x) - x;
```

Suppose that you have called your file `map`, then start GRIND by typing `grind map` and wait until you get

```
 G R I N D (94.10)  model analysis:
```

Now enter the following lines

```
option map
option pos
terminal x11
r=3.3
x=0.1
finish 20 20
run
timeplot
```

Which tells GRIND that you want this model to be a map, which prevents negative values of the `x` variable, which tells GRIND that you are on an X-terminal, which sets parameters and the initial condition, which asks for twenty time steps, calls for a run and plots the data. You can draw the famous bifurcation diagram of the logistic map by the `bifurcate` command. First define a two-dimensional space

```
r=1
axis x r 1 4
axis y x 0 1
erase
2d
```

Then type

```
 bifurcate x 4 200 100 20
```

which says that you want to run along the x-axis until $r = 4$. For every new value of r the map is integrated for 200 time steps to eliminate transients. We simulate the map for one hundred different values of r and we plot 20 data points for every value of r. Most commands for differential equations also apply for maps; the `bifurcate` command however has a different meaning for differential equation models. We now run the model for a thousand time steps, and plot the results in a Takens reconstruction

```
axis t x 0 1
finish 1000 1000
run
erase
takens 1
erase
takens 2
```

where the `axis` command defines the axis of the Takens plot. The first `takens` command plots x_{t+1} as a function of x_t, and the second one plots x_{t+2} as a function of x_t. Now we do the same with some noise on the growth rate r

```
noise r 3.5 0.1
run
takens 1
```