G R I N D: tutorial

Rob J de Boer & Ludo Pagie

Theoretical Biology, Utrecht University
Email: R.J.DeBoer@uu.nl

This document accompanies GRIND Version 2.12

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.

Model definition. Use any text editor (e.g., xemacs, nedit, WordPad) to define your model by writing a plain ASCII file of the following format

```
F = R^n/(h^n + R^n);
R '= r*R*(1-R/K) - b*N*F;
N '= b*N*F - d*N;
```

Which defines a predator-prey model with a sigmoid functional response. The model has three equations, where `F` is an algebraic variable, `R` and `N` are variables (for prey and predator), and `n, r, K, b` and `d` are parameters.

Note that algebraic expressions, here `F`, have to be defined before the differential equations. The GRIND preprocessor will also understand you if you use C-functions like `sin(), cos(), log(), exp(), sqrt(), fabs(), mod(x,y), max(x,y)` and `min(x,y)`.

Model analysis. When GRIND is installed properly you start GRIND in Unix by typing `grind model` where we assume that the file that you made above is called `model` or `model.grd`. In case you are using Windows you have to use this `.grd` extension (which should be linked to the `grind.bat` program), and start GRIND by double clicking the `model.grd` icon. GRIND will check if your model has been preprocessed before, and will start up its command line driven interface. After this you just start typing commands like

```
r=1;K=1
b=.25
h=.1;d=.1
n=2
R=0.1
N=0.1
parameters model.txt
```

which initializes all parameters, the initial condition, and gives an overview of all parameter values. Because the `parameters` command contains a filename (`model.txt`), the parameter values are also saved in that file.

Later you can read the old parameter values with `read model.txt`. In fact you can read files with all sorts of GRIND commands. You can stop GRIND with the `bye` command.

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

```
finish 20 20
run
axis v 0 2
timeplot
```

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. The `axis v` 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`!

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

```
axis x R 0 1
axis y N 0 2
2d
nullcline R N
vector
```

which defines a state space by two axis commands. The horizontal "x" axis is defined by the prey `R`, and runs from `R=0` to `R=1`. The vertical "y" axis is defined by the predator `N` and runs from `N=0` to `N=2`. The `2d` command displays the phase space on the screen, in which nullclines are drawn and a vector field is indicated.

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
e2n
grid
```

where `e2n` is a predefined macro for `erase;2d;nullcline`, and `grid` draws a grid of trajectories.

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 R
```

Note that we prevent `h=0` by starting the Z-axis at `h=0.001`.

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 eigenvalues of the Jacobian matrix of the equilibrium point.

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

2d	make a 2-dimensional phase space
axis	define the identity and scaling of an axis
bye	leave GRIND
bifurcate	make a series of Poincaré sections
continue	follow a steady state as a function of a parameter
cursor	ask for a mouse click to set the initial condition
display filename	show the contents of a text file
eigen	compute eigenvalues
finish	set the time span and number of points reported
grid	start many trajectories
help	get help
keepvar	copy the final state into the initial state
newton	approach a close-by equilibrium point
noise	set white noise on a parameter
nullcline	draw nullclines
option	various options
parameter	list parameter setting and/or save them into a file
poincare	make a poincaré section
read filename	read a file with parameters and/or commands
run	solve the model numerically
timeplot	depict the solutions
vector	show the vector field
where	where am I?

Table 1: A sample of the most important GRIND commands. Note that GRIND commands can be abbreviated to the first two letters.

and continue that equilibrium along that axis. For instance

```
newton
axis x K 0 3
axis y N 0 5
2d
continue x
```

which plots the equilibrium value of the predator `N` for various values of the carrying capacity K.

White Noise. You can set noise on a parameter by typing `noise r 1 0.1 [pos];` which gives the parameter r an average of one and a standard deviation of 0.1. If the `pos` option is provided, negative parameters are set to zero. 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 r off` or use `noise off` to switch off the noise on all parameters.

Help, 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, your current graphics screen can be saved in PostScript files by `export file`, which creates a file "file.eps" with (Encapsulated) PostScript output. GRIND has a help facility which you activate by typing `command help`, where `command` is the name of the command you need some information about. The current status of a command is printed with `command status`. For detailed information you will have to consult the GRIND manual in GRIND's `doc` directory.

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.grd`, then start GRIND by typing `grind map`, and enter the following lines

```
option map
option pos
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 sets `r=3.3` and the initial condition `x=0.1`. Then ask for twenty time steps, call for a run and plot the data.

You can draw the famous bifurcation diagram of the logistic map by the `bifurcate` command.

```
r=1
fin 100 100
run
axis x r 1 4
axis y x 0 1
2d
bifurcate x 20
```

First run to the attractor for `r=1` and set a longer simulation time. Next define a two-dimensional space, in which you plot 20 points for several values of r.

Most commands for differential equations also apply for maps; the `bifurcate` command will do Poincaré sections with differential equation models, however.

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
takens 1
takens 2
noise r 3.5 0.1
run
takens 1
```

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. Finally, do the same with some noise on the growth rate r.

Enjoy!