# G R I N D

## GReat INtegrator Differential equations

**Rob J. de Boer & Ludo Pagie**

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

*This document accompanies GRIND Version 2.13*

GRIND is a user friendly computer program for the study of differential equation models by means of numerical integration, steady state analysis, and phase space analysis (e.g., nullclines and separatrices). The model equations are defined in a very natural formalism, and one can easily change parameters and initial conditions. The user interface to GRIND is based upon a simple command language. GRIND supports interactive OpenGL and/or X11 graphics, and exports Postscript files.

The best way to learn more about GRIND is to walk through the example on page 11, and/or try the tutorial. Afterwards read the rest of the manual to see what else GRIND has to offer to you. GRIND commands have a help function to remind you of their syntax.

# 1 Model

A model consists of a set of differential equations and possibly algebraic expressions. Differential equations are defined in the notation `variable '= equation;`. For example, the logistic equation would be `x '= r*x*(1-x/K);` A good example of an algebraic variable would be a quasi steady state variable, or a function appearing repeatedly in the model.

For example, the predator-prey model on page 11 is

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

Which has three equations, where `F` is an algebraic variable, `R` and `N` are variables (for prey and predator, respectively), and `r, K, b` and `d` are parameters.

Please note the semicolons (;) at the end of each equation. Each equation may therefore cover several lines. The algebraic expressions have to precede the differential equations, and an expression should be defined before it is used in another expression. The names of variables, algebraic variables, and parameters should be less than 11 characters long, and names start with a letter. The other nine characters should be letters or digits. Equations consist of names, operators (i.e., `+ - * / ^`), numbers, and/or functions. Numbers in equations, e.g., 3.14159265, should also be shorter than 11 characters. You can use all standard C-functions with one or two arguments. Examples are `exp( ), log( ), sqrt( ), sin( )`, and `fabs( )`. We have added for you `max( , ), and min( , )`.

For some functions GRIND will tell you that she doesnot know how to compute the derivative. In such circumstances GRIND will give you a warning, and will suppress the Jacobian matrix. (This matrix is required for variable time step integration with ROW4A and for neighbourhood stability analysis).

## 2 The basic idea

A model has to be typed into a text file. You may use any ASCII text processor (like emacs or WordPad) to create such a file. In Windows you should save this file with the `.grd` extension. For Linux users no such extension is required, but if you like extensions also use `.grd`. Linux users should start grind by typing `grind filename` (whether or not the file has the `.grd` extension). Windows users simply double click the `filename.grd` icon. This activates the GRIND preprocessor that will read your file, will report possible errors, and will create a C-program. This C-program is compiled, loaded with the GRIND libraries, and is executed. When you get the `GRIND>` prompt, you can start typing GRIND commands. The first thing that you may want to do is to initialize your parameters (e.g., `b=3`), and your variables (e.g., `x=1`). It is usually a good idea save initializations in a file, e.g., `parameters filename` or `parameters model.txt`, and to start your next session with reading this file (e.g., `read filename`). Whenever a computation takes too long, you may try to regain control by hitting `Control C` key.

Subsequently, one probably wants to see the phase space by calling the `2d` or `3d` commands. The axes of the phase space can be changed with the `axis` command. Having the model and the phase space fully specified you may start analyzing the model in various ways. Start numerical integration using `run`, draw the x and y nullclines (e.g., `null X Y`), try to find an equilibrium using `newton`, etcetera. When you are done you can stop GRIND using the `bye` command.

GRIND commands may be abbreviated to the first two letters. Most commands may have several arguments that typically have some sensible default. GRIND usually remembers previous arguments. For example, after having asked for the $x$ and $y$ nullclines by `nu x y` (for `nullcline x y`), the next time you get the same nullclines by just typing `nu` without arguments. GRIND has a reasonable help funtion: `command help` explains the meaning and syntax of the command. `help` lists all possible commands. When using the `OpenGL` graphics interface you can rotate 3-dimensional phase space with the left mouse-button, zoom in and out with the `z` and `Z` key, and reset the 3-dimensional projection with the `r` key.

Numerical integration starts when you give the `run` command. The time stretch that the integrator will compute is called the `finish`. Several times during this time stretch the integrator is interrupted for generating output. The number of intervals at which outout is generated is set by the second argument of the `finish` command. The interrupts at each interval may be used to introduce random noise, and play a role in Poincaré sections. GRIND either uses OpenGL or X11 for the graphical display. Graphics can be exported to Postscript or TEX by the `export` command.

In GRIND we speak of the "'attained state" and the "initial state". The initial state corresponds to the values that you have typed. The attained state corresponds to the values numerically attained by GRIND (e.g., by integration or by a Newton-Raphson iteration). The attained state can be copied into the initial state by the `keepvar` command.

# 3 Reference manual

Generally, GRIND input lines are of the following form:

`command argument1 argument2 ... argumentn;`
Thus, different arguments are to be separated by a space " ". Arguments may be integer or real values, strings, characters, and keys like `off` or `log`. Several commands may be placed on one line if they are separated by semicolons (;). When commands are positioned on different lines you may omit the semicolon. Parameters and variables can be set by simple assignments: `identifier = value;`. Again the semicolon (;) may be omitted when you type a newline. In this manual a "/" in this manual means "or", i.e., a "/" separates alternatives. For example, `axis x/y/z/t` means that the `axis` command expects `x`, `y`, `z`, or `t` as its first argument. Second, when arguments have square brackets "[ ]" around them this means that those arguments can be omitted.

Here are all possible GRIND commands in alphabetical order:

`variable = expression; parameter = expression;`
This sets parameters and initial conditions. The expression is usually just real or integer number. One may use expressions involving the operators `+-*/^`, and the names of variables and parameters.
See: `calculator;`

`2d;`
A 2D state space is generated.

`3d;`
A 3D state space is generated.

`axis x/y/z/t variable/parameter minimum maximum [nticks] [log];`
One axis of the state space is defined, i.e., is related to the variable or parameter appearing as the second argument. The axis runs from `minimum` to `maximum`, and has `nticks` tickmarks. The `log` keyword specifies that the axis should be made logarithmic. A logarithmic axis runs from $10^{minimum}$ to $10^{maximum}$. `axis t` defines the axis of the plot generated by the `takens` command.
Default: `axis - - 0.0 1.0 3;`

`axis h/v minimum maximum [nticks] [log];`
Defines the horizontal or vertical axis of time plots. The arguments are the same as those explained above. The only difference is the absence of a variable name. The horizontal axis of a time plot can be changed but is automatically reset to length `finish` by the `run` command.

`bifurcate x/y/z [npoints [nsteps [transient]]];`
This is a brute force method of making a bifurcation diagram. The axis indicated by the first argument should correspond to a parameter. This parameter is changed in `nsteps` small steps along the axis. For each parameter value the model is run for `transient` time steps, after which `npoints` Poincaré crossings are recorded. The starting point of the bifurcation diagram is the current value of that parameter. You should be sure that for that parameter value the trajectory crosses the Poincaré plane within the normal `finish`. The waiting period can also be used for adjusting the location of the Poincaré section: if `option adjust` is set, GRIND will use the transient to estimate an average value for the location of Poincaré section. Finally, in combination with `option map`, GRIND will not do Poincaré sections but will iterate your model for `npoints` time steps (after a possible waiting period of `transient` time steps).

SEE: `option adjust/keepvar/map, poincare`.
DEFAULT: `bifurcate - 1 50 finish;`

`bye;`
GRIND will stop, close files, and graphic windows.

`calculator expression;`
Prints the outcome of expressions involving the operators `+-*/^`, some functions, and the names of variables and parameters. A variable name refers to the initial state of that variable.

`continue x/y/z [arclength];`
After having attained an equilibrium by the `newton` command you may want to continue the equilibrium along some parameter. There are sophisticated packages that do this sort of bifurcation analysis for you. GRIND implements just a very primitive continuation algorithm. The basic idea is to have a parameter on some axis of your phase space, and to `continue` along that axis (see the first argument `x/y/z`). The `arclength` argument influences the maximum arclength of the continuation algorithm. This should just be a relatively small number.
DEFAULT: `continue x 0.01;`
SEE: `newton, perturb`.

`cursor;`
Activates the cursor in order to select a new initial state. Move the cursor to the place in the 2-dimensional phase space that you want to select, and click your mouse. The point will be copied into the initial state.

`display file [pattern];`
GRIND display the contents of the file appearing as the first argument. If you provide a second text argument, only those lines of the file containing that text will be listed.
DEFAULT: `display model;`

`e2n;`
Executes the commands: `erase;2d;nullcline`.

`e2r;`
Executes the commands: `erase;2d;run`.

`e3n;`
Executes the commands: `erase;3d;nullcline`.

`e3r;`
Executes the commands: `erase;3d;run`.

`ert;`
Executes the commands: `erase;run;timeplot`.

`eigen [jac];`
Performs a neighborhood stability analysis, giving you the eigenvalues and eigenvectors of the Jacobian matrix. The Jacobian is calculated for the state attained by the last `newton` command. The argument `jac` lets the jacobian matrix be printed.
SEE: `option mark, perturb`.

`erase;`
The screen is cleared, a new plot is started, all graphics segments are deleted.
SEE: `ert, e2r, e2n, e3r, e3n`.

`export filename[.ext];`
Exports the current graphics window into the name file. The extension determines the graphics format: the default is Encapsulated Postscript (`.eps`). If you are using the OpenGL environment, you can also export PDF-files (`.pdf`) and TEX-files (`.tex`). We are working on PNG-files.
DEFAULT: `export model.eps;`

`finish time [nintervals];`
`finish time` sets the time stretch of the integration. The integrator is interrupted `nintervals` times for generating output (and/or for random noise).
DEFAULT: `finish 100.  10;`

`format [print/]eigen/vector/axis [ndigits];`
GRIND prints numbers with the C "`%-w.ng`" format where `w` is the width of the field, and `n` the number of digits to be printed to the right of the decimal point. The `ndigits` arguments the precision with with numbers are printed generally (`print`), for eigenvalues (`eigen`), for eigenvectors (`eigen`), and along for the axis of the phase space (`axis`).

`grid [npoints];`
Generates a grid of trajectories in a phase space. Trajectories start at regular intervals in the state space. The `npoints` argument defines the number of starting points per row. `grid` is influenced by several options. The starting points of trajectories are shown as nice little arrows when you provide `option initial`. Trajectories will also use the axes of the phase space as starting points if have set `option axis`.
SEE: `finish, option axis.`
DEFAULT: `grid 5;`

`help;`
Lists all possible commands. Get help on a command by `command help`, and get the current state of the command with `command status`.

`integrator row4a/rk/euler [tolerance];`
This selects the algorithm for numerical integration. First provide the name of the algorithm (where `rk` stands for Runge Kutta). For the variable time step integrators, the optional `tolerance` argument sets the tolerance. In the case of Euler integration, this argument sets the time step `delta`.
DEFAULT: `integrator row4a 0.0001;`

`keepvar;`
The attained state is copied to the initial state, i.e., all variables obtain as the initial value the value they attained in the previous integration.

`load file;`
Reads a file containing the output from a file that was generated by a `run` command. In fact `load` will read any file of `nvar+1` columns, where `nvar` is the number of differential equations, and the first column is assumed to be the `time`. `load`ed data can be replotted in a phase space by the `plot` command or in the time plot by the `timeplot` command. The first row of the `load`ed file becomes the new initial state and the last row becomes the new attained state.
SEE: `plot, run`.

`macro [name command;command;command];`
Several commands can be combined into a named macro. The first argument is the new name of the set of commands. They are all listed by an empty `macro` statement. A macro is undefined by `macro name`.

`mine [any argument];`
This allows for the execution of a C-function supplied by the user. This of course requires a little knowledge of how GRIND is programmed. In the file created by the preprocessor you will find a dummy function `mine(argc,argv)` that simply echoes the arguments on the command line. If you want to add something to GRIND just modify this routine.

`mode [alpha/2d/3d/none];`
In graphics mode (i.e., 2d or 3d mode) the output of integrations and continuations is immediately plotted in the phase space. GRIND assumes graphics mode automatically after a `2d` or a `3d` command. GRIND resumes alphanumerical output if the mode is set accordingly, i.e., by the `alpha` argument. `mode none` generates no visible integration output at all (this may be useful in combination with `timeplot`).
DEFAULT: `mode alpha;`

`nep;`
Executes the commands: `newton;eigen;perturb`.

`newton [tolerance];`
Searches an equilibrium using Newton-Rhapson's method with an optional `tolerance`. The initial state is used as an initial guess, the result is stored in the attained state. The attained state may be continued with `continue`. Invariant manifolds of the attained state can be studied with `perturb`.
SEE: `continue, eigen, nep`.
DEFAULT: `newton 1e-9;`

`noise parameter mean stdev [off] [positive];`
Sets random noise on the listed parameter. The noise is set each time the integrator is interupted for output (which is determined by the second argument of `finish`). Noise resets the parameter value to random numbers drawn from a normal distribution with the `mean` and `stdev` given as arguments. `off` switches the noise off. To prevent negative parameter values use the `positive` argument.
DEFAULT: `stdev = 10% of mean;`
SEE: `finish`

`nullcline [variable ... variable];`
`nullcline` draws 0-nullclines for each of the variables.

`option ...[off];`
Sets an option:
`adjust`: adjust the location of the different Poincaré planes in the `bifurcation` command.
`axis`: `grid` points include the axes of the phase space.
`backward`: integrate backwards in time.
`calculator`: variables in `calculator` use their initial or final value.
`closed`: the frame of the box shown in 3D spaces is closed.
`erase`: the screen is erased when `timeplot`, `2d`, or `3d` is called
`initial`: GRIND graphically indicates the integration starting point.
`keepvar`: in `bifurcate` the next sample continues at the point attained in the previous integration.
`label`: labels an axis with ticks and labels.
`logfile`: stores commands in a `model.log` file.

`map`: GRIND assumes that you are studying difference equations (instead of a differential equations). This influences `bifurcate` and eigenvalue calculations.
`mark`: marks the steady state and its stability whenever `eigen` is called.
`points`: draw single points instead of connected lines as integration output.
`positive`: negative variables are set to zero.
`truncate`: trajectories are stopped when they leave the phase space. This is very helpful in combination with `backward` integration.

An option is switched on by `option name` and off by `option name off`. Use `option status` to see the current values of all options.

`output [+-] variable variable expression parameter;`
This opens an output buffer that containing the variables and/or algebraic variables to be printed or plotted as the output of the numerical integration. By default all variables are contained in the output buffer. Using the `+` or `-` option anywhere between the names causes items to be added or deleted from the buffer.

`parameters [file];`
`parameters` provides a listing of the parameter setting. Such a listing can be saved on a file by providing an filename. Using the `read` command you can read such a parameter listing again.

`perturb vector [step];`
This allows you to study one-dimensional stable and the unstable manifolds of an equilibrium. Assuming that you have attained an equilibrium (by `newton` or `run`) for which you have already asked the eigenvalues and eigenvectors (by `eigen`), `perturb` makes a little perturbation of size `step` along the `vector`$^{\text{th}}$ eigenvector. The result is copied into the initial state. Thus, a subsequent `run` starts along this particular eigenvector. The `vector` argument corresponds to the eigenvalues provided by `eigen`, the default (`vector=1`) being the eigenvector corresponding to the largest eigenvalue. GRIND is unable to generate the 2D outset that is spanned up by a pair of imaginary eigenvectors. You will get an error message if you would attempt to do this. When you `perturb` along a stable manifold (i.e., along a separatrix), `option backwards` is automatically switched on. It is advisable to also switch `option trunc` on in such circumstances. The second `step` argument specifies the distance to the equilibrium. This should typically be a small number. If the `step` argument is negative, the perturbation is in the opposite direction along the eigenvector.
DEFAULT: `perturb 1 0.0001;`
SEE: `eigen, newton, nep, option trunc`.

`plot;`
Replots a previous trajectory in a 2d or 3d phase space.
SEE: `load`.

`poincare variable value [up/down/both];`
This selects the hyperplane, `variable=value`, of the Poincaré section. With `up/down/both` you can select crossings for the variable increasing (`up`), decreasing (`down`), or both directions. Poincaré sections are influenced by the `finish` and its number of interval.s For each point on the Poincaré section GRIND runs the numerical integrator for `finish` time steps. Each interval a check is made for a cross of the Poincaré section. When a crossing is found GRIND will iterate the numerical integration until the crossing is accurately determined.
SEE: `finish`.
DEFAULT: `poincare variable 0 up;`

`poincare npoints [tolerance];`
This attempts to record `npoints` crossings of the trajectory through the Poincaré section. The second `tolerance` argument determines the accuracy with which the crossing is determined. The initial state is used as the starting point for the trajectory until the first crossing. Each succesful point that is found will be the initial condition for the next point. Note that for each point on the Poincaré section GRIND will maximally run for `finish` time steps.
SEE: `finish`.
DEFAULT: `poincare 1 0.0001;`

`rgb number real real real;`
Set the Red-Green-Blue values of the color `number`. Use `rgb status` to see the current values.

`readfile filename;`
Commands are read from a file.
DEFAULT: `readfile model.txt;`

`run [file];`
Integration starts. In graphics mode the output is plotted on your screen, in alphanumeric mode (`mode alpha`) the output is printed on your screen. If a filename is provided, the integration data is saved on a file. Such a file can be read again by `load`.
SEE: `finish, load, option points, timeplot`.

`shade [x/y/z] variable [integer];`
The nullcline surface of the `variable` is shaded along the the x, y, or z-axis. The `integer` argument specifies the number of lines.
DEFAULT: `shade z variable 10;`

`takens [delta];`
Generates a Takens reconstruction of the variable set by the `axis t` command. This means that the variable is plotted at a function of itself `delta` time steps ago. Thus, `takens` plots a variable at time `t` as a function of the same variable at time `t-delta`. This is a standard technique for studying chaotic time series.
DEFAULT: `takens 1;`

`timeplot [variable ...variable];`
A time plot of the previous integration is generated. Thus, you have to do a `run` before you can call `timeplot`. The horizontal axis runs from zero to `finish`. The vertical axis is to be defined by `axis v` command. The variables to be plotted are tyically defined by the `output` buffer. You may select a subset of them by supplying names as arguments to the `timeplot` command.
SEE: `load, run`.

`triangle [size];`
Plots a triangle denoting the vector field at the initial state. The `size` argument scales the size of the triangle.
DEFAULT: `triangle 1;`

`value variable parameter;`
Prints the values of parameters, algebraic variables, and/or variables.

`vector [npoints];`

Draws a vector field. The `npoints` argument defines the number of arrows per row.
DEFAULT: `vector 5;`

`where [initial] [file];`
The attained state (or the initial state with the `initial` option) is indicated in the state space, and is printed with the derivatives. When you provide a filename as the last argument the state is also written to that file. Such a file can be read by the `read` command.

# 4 Credits and Algorithms

Starting in 1983, Rob de Boer developed GRIND as a successor of the TRICLE system devised by P. Hogeweg and B. Hesper ("Interactive Instruction on Population Interactionbs" *Comp. Biol. and Med.* 8, 319–327, 1978).

GRIND implements three different algorithms for numerical integration. The default is the variable-time-step ROW4A method which was developed by B.A. Gottwald & G. Wanner for the analysis of stiff systems of ordinary differential equations ("A Reliable Rosenbrock Integrator for Stiff Differential Equations" *Computing* 26, 355–360, 1981). ROW4A requires a Jacobian matrix which is default provided by the preprocessor. The second integrator is the variable-time-step Runge Kutta method provided by W.H. Press, B.P. Flannery, S.A. Teukolsy & W.T. Vetterling in *Numerical Recipes*. The third is a simple (fixed-time-step) Euler intergrator. The integrator is set interactively with the `integrator` keyword.

GRIND searches for nullclines by stepping through 2d sections of the phase space keeping track of switches in the sign of the derivatives. For Poincaré sections GRIND checks each interval of a trajectory (see `finish`) whether it has crossed the Poincaré plane. Subsequently, the actual point is determined by a Newton-Raphson iteration.

GRIND finds roots of differential equations by the Newton-Raphson algorithm supplied by *Numerical Recipes*. Eigenvalues and eigenvectors are computed by means of the EISPACK routines (B.T. Smith *et al., 1976, Lect. Notes in Computer Science*, Springer-Verlag, Berlin).

Ludo Pagie translated GRIND into the C programming language and made the OpenGL interface. GRIND processes the commandline using the "getline" software from the NFTP package (`http://www.ncftp.com/`. This code originates from Chris Thewalt (`thewalt@ce.berkeley.edu`) and is updated by Mike Gleason (`http://www.NcFTP.com/contact/`)

The rotations of the graphics under GL is based on the "trackball" algorithm, originally adapted (ripped off) from `projtex.c` (written by David Yu and David Blythe). See the SIGGRAPH '96 Advanced OpenGL course notes. The compiler environment for the windows version, `MinGW`, is uncopyrighted (see: `http://www.mingw.org`). In addition, we use the "pthread-win32" library (`http://sources.redhat.com/pthreads-win32`), and the "glut" library (`http://mywebpage.netscape.com/PtrPck/glut.htm`).

# 5 Examples

These examples can all be copied to your own directories from the GRIND `models` subdirectory.

## 5.1 a Predator Prey model

Suppose you have the text file called `monod.grd` as displayed on the left. This contains the differential equations of a predator-prey interaction with a saturated functional response.

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

The algebraic expression `F` defines the functional response, and is used in the differential equations `R'=` and `N'=`.

Under Windows you can make this file with WordPad, in Unix with nedit or xemacs. In Windows you double click on the `monod.grd` icon, in Unix you type `grind monod`.

Consider the session given below, where the left column contains what you type:

```
r=1;K=1
b=0.25; d=0.2; h=0.1
parameters monod.txt
```

This sets the parameters. They are listed and stored in a file called `monod.txt`.

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

Set up a phase space with $R$ on the horizontal and $N$ on the vertical axis, and draw their nullclines.

```
finish 100 100
R=0.1;N=0.1
run
erase
axis v 0 2
timeplot
```

Set the integration time to 100 timesteps, and print 100 steps. Set the initial condition `R=0.1;N=0.1`, and run the integrator. Plot the output with a vertical time plot axis between zero and two.

```
e2n
R=0.5;N=1.5
newton
eigen
```

The nullclines are redrawn, and from the initial guess `R=0.5;N=1.5` we find the non-trivial steady state, and its stability.

```
ax x K 0.01 2
2d
continue x
```

This state is continued for the parameter `K` along the X-axis.

```
ax x R 0 2
ax z K 1 2
e3n
shade R
```

Finally we set op a 3-dimensional phase space and shade the nullcline of the prey. If you are using `OpenGL` you can click the left-button of your mouse for rotation.

Note that if you repeat a command (see the repeats of `nullcline` in `e2n` and `e3n`) you don't have to

repeat all arguments.

### 5.2 An example with a macro

Consider the following immune network model [De Boer & Hogeweg 1989, *Bull. Math. Biol. 51: 381-408*]. The column on the left is the model, which is a file called `bc.grd`, and that on the right is a text file called `bc.txt`.

```
h1= a*x2;
h2= a*x1;
f1= p * h1/(p1 + h1) * (p2/(p2 + h1));
f2= p * h2/(p1 + h2) * (p2/(p2 + h2));
x1'= s + x1*(f1 - d);
x2'= s + x2*(f2 - d);
```

```
a=1; p=1.5
d=1; s=10
p1=1000; p2=1E+06
x1=1e8; x2=1e5
fin 100 100
ax x x1 -1 8 4 log
ax y x2 -1 8 4 log
ax z a  -3 0 4 log
ax v -1 8 log
macro cne cursor;newton;eigen
```

In GRIND you type:

```
read bc.txt
2d
nullcline x1 x2
run
e2n
grid 5
```

This starts with reading the `bc.txt` file, and plotting nullclines. Then run the integrator. Finally nullclines are redrawn, and a `grid` of trajectories is computed.

```
cne;cne;cne
```

Call the macro three times, and click three times close to a steady state to check its stability from the eigenvalues.

```
e3n
shade x1
bye
```

Three-dimensional nullclines are draw, and `x1` is shaded.

### 5.3 Models that depend on time

For seasonal variation on a parameter one should count days by writing a differential equation `day '= 1;`. Taking the $\sin[2\pi(\text{day} - \Delta)/365]$ one obtains a yearly seasonal variation between $-1$ and 1, with a parameter $\Delta$ to shift the peaks of the sinusoidal function to the appropriate time of the year. For instance, to set seasonal variation on the growth rate $r$ in the above-mentioned predator prey model one would add an algebraic expression like:

```
r = r0 + e*(sin(2*3.1416*(day-Delta)/365));
day '= 1;
R '= r*R*(1-R/K) ....
```

where the $e$ parameter determines the amplitude of the variation of $r$ around $r_0$.

To make a seasonal variation that is "on" during one period of the year, and "off" the rest of the year, one can truncate the sinus function using only its positive part, e.g., `S=max(0,sin(...))`. To make a function that switches on and off at time points defines by the user, one could multiply two sinus functions, that are appropriately shifted in time, and take this product through a Hill-function to let it switch between zero and one with arbitrary steepness.

For example, consider a seasonal insect population that is active from April until June, e.g., from day 100 to 200. During this season the eggs (`E`) hatch at rate $h$ per day. During their season hatched adults survive with a probability depending on the adult density. After the season they lay eggs and die rapidly at rate `m` per day. Look at the model called `season.grd` with runfile `season.txt`:

```
on = max(0,sin(2*3.14*(day-Ton)/365));
off = max(0,sin(2*3.14*(day+182.5-Toff)/365));
S = (on*off)^5/(eps^5+(on*off)^5);
day '= 1;
A '= h*S*(E - d*A*(1+e*A)) - m*(1-S)*A;
E '= n*(1-S)*A - h*E*S - l*E;
```

```
Ton=100
Toff=200
eps=0.01
fin 730 365
out S
ma go keepvar;ert
```

```
re season.txt
run
timeplot
```

Check that the expression `S` indeed switches on around day 100 and off around day 200.

```
d=0.001
e=10
h=1
l=0.001
m=1
n=10
E=1
axis v 0 10
output S A E
run
ert
go
go
```

Set parameters such that there is strong density dependent mortalility of the adults during the season (`e=10;d=0.001`), and that adults die rapid when the season is over (`m=1`). Eggs hatch rapidly, and decay slowly. Each adults lays `n=10` eggs per day at the end of the season. Rescale the vertical axis of the timeplot and ask for output of `S A` and `E`. Start with one egg, and call the `go` several times until the attractor is approached.

## 5.4 Models with a second derivative

Consider Duffing's equation, and Udea's analysis of it, discussed in the highly recommendable book *Nonlinear Dynamics and Chaos* by J.M.T. Thompson & H.B. Stewart (Wiley, London, 1986):

$$x'' + kx' + x^3 = B\cos(t)$$

By setting $y = x'$ this second-order differential equation translates into the first-order GRIND model listed below. Suppose it resides in the file `duffing` in your directory. Consider the following session, where $t$ defines the time, `y` refers to $x'$, and `x` defines $x''$. The column on the left is the model (called `duffing.grd`), and that on the right is a file called `duffing.txt`.

```
t '= 1.0;
y '= b*cos(t) - k*y - x^3;
x '= y;
```

```
b=7.5; k=0.05
x=3; y=4
axis v -3 3
axis x x -4.5 4.5
axis y y -9 9
axis z t 0 3
finish 31.41 100
output x
```

```
re duffing.txt
2d
run
ert
```

Read the input file, make a 2-dimensional phase space, run, and make a time plot.

```
finish 6283.18 1000
axis x x 1 4
axis y y -6 6
2d
option points
option initial off
run
export fig.eps
bye
```

Resize the phase space, switch the indication of the initial state off, and plot a stippled trajectory. Export the graphics window to an encapsulated PostScript file `fig.eps` that can later be printed.

### 5.5 Maps

GRIND can be fooled to analyse difference equations or maps. 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 in a file as `x '= r*x*(1 - x) - x;` In GRIND you would type

```
option map
option pos
r=3.3
x=0.1
finish 100 100
run
timeplot
```

Which tells GRIND that you want this model to be a map, and where `option positive` prevents negative values of the `x` variable. Set the parameter and the initial condition, and run the model for 100 time steps.

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

```
axis x r 1 4
axis y x 0 1
erase
2d
r=2;run
erase
2d
bifurcate x 10
```

Define a two-dimensional space with the `r` parameter on the horizonal axis. Set the growth rate to an intermediate value and run the model to approach the attractor. Then plot a 10 points of the trajectory for fifty different values of `r`.

Note that if you use `bifurcate` without `option map` it will make many Poincaré sections for you.

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
```

# 6 Running and installing GRIND

GRIND is available from `theory.bio.uu.nl/pub/rdb` and `theory.bio.uu.nl/rdb/grind.html` provides some installation instructions.

## 6.1 Unix

The GRIND system consists of a script called `grind`, the preprocessor `grindpp`, and the library `libgrind.a`. During installation the script learns where the library and the preprocessor are installed. Thus, after installation a user has to have the `grind` script somewhere in the Unix `path`.

Run your models by `grind [-jx] filename`. GRIND will open a graphics window as soon as you start doing graphics. Graphics can be exported as (encapsulated) Postscript, TEX-code, and PDF files. You can suppress the Jacobian with the option `-j`, and ask for X11 graphics with the `-x` option. Options can of course be combined (i.e. `-xj` and `-jx` are both correct) and be omitted.

For installation check the first lines of the makefile in the GRIND directory. Here you may have to set the `X11DIR` variable, which gives the location of the X11-library. It is preset to the Linux location (`X11DIR = /usr/X11R6/lib`), and on Sun Solaris this should be `X11DIR = /usr/openwin/lib`.

## 6.2 Windows

Type your model as a `text-only` file using any text editor (e.g., `WordPad`). Give your models the `.grd` extension (which is still free). The first time you open a model you select the `Grind\bin\grind.bat` file as the program to open `.grd` files.

GRIND uses "minimal GNU for Windows", and after a typical installation the required `MinGW` files and the GRIND files are stored in a folder called `Grind`. This folder is distributed as the `grind-2.??.zip` file, which you can unpack anywhere in your system (e.g., in `My Documents` for single users, and higher up for multiple users). After unpacking the `zip`-file, run the `setup.bat` program in the `Grind` folder. This makes the script `Grind\bin\grind.bat`. Double click `model.grd` in the `Grind` folder to define this `grind.bat` as the default program to open files with the `.grd` extension.