user's guide

iDMC

interactive Dynamical Model Calculator

available at www.dss.uniud.it/nonlinear[1]

[1]Last revised 26 April 2005. A special thanks to Daniele Pizzoni, Alexei Grigoriev and Gianluca Gazzola for their help in various aspects of the guide's preparation. As usual the author, Marji Lines takes all responsibility for errors. Please address comments, criticisms, corrections and suggestions to lines@dss.uniud.it

GNU notice

contents

1. Main menu

File

a. `New model` Clicking on this button brings up the directory `models` from which one is chosen to be opened.

b. `Close model` Clicking on this button closes the current model and all related plots.

c. `New plot` This is the heart of the program. Clicking on this button brings up the menu for plot types, that is, the menu from which the user chooses the routine to be applied to the opened model. The options available for the class of model selected (basically, whether time is continuous or discrete and how many dimensions there are to the model system) will be highlighted as you drag the mouse over the plot name. See Sections 3-9 for detailed descriptions of specific routines.

d. `Close plot` Clicking on this button closes the current plot.

e. `Save image as` This button allows the user to save a plot to a file. Clicking on this button from a plot produced by one of the routines brings up the window in which the directory and file name are chosen. Plots are saved as .png files.

f. `Quit` Clicking on this button closes the current window.

g. `Exit` Clicking on this button closes the session and exits the program.

Command

The composition of the `Command` menu varies with the routine.

a. `Start` Clicking on this button resets ranges and starts the algorithm.

b. `Stop` Clicking on this button stops any calculation. This is particularly useful if a long calculation has been configured incorrectly.

c. `Clear` Clicking on this button clears the current plot.

d. `Redraw` Clicking on this button allows the user to redraw the same plot with newly defined ranges.

e. `Reset` Clicking on this button releases the start button or menu.

Plot

The composition of the `Plot` menu varies greatly with the algorithm and the details are given in the next section under the routine type.

Options

This menu permits the user to change certain plot characteristics from the default. The `Big dots` and `Connect dots` options are available only for some discrete-time routines.

a. `Size` Clicking on this button opens a menu giving the following choices
`500×500`
`600×600`
`Fit to window`
`Custom size` This button opens a window in which the plot dimensions are fixed.

b. `Show axes` Clicking on this button opens a menu giving the following choices
`Both`
`None`
`Domain only`
`Range only`

c. `Colors` Clicking on this button opens a menu giving the following choices
`Chart background`
`Plot background`

d. `Transparency` Clicking on this button makes it possible to see points plotted beneath other points, giving an idea of how often that point is visited.

e. `Transparency factor` Clicking on this button opens a window in which it is possible to change the transparency factor (which varies over (0,1) from the default (0.2) to increase transparency (smaller number) or decrease transparency (larger number).

f. `Gridlines` Clicking on this button produces gridlines on the plot.

g. `Big dots` Clicking on this button magnifies the dots on the plot. This option is useful especially when studying trajectories and orbits of discrete-time systems.

h. `Connect dots` Clicking on this button produces a plot in which the points plotted are connected, again useful in studying discrete-time systems.

i. `Crosshair` Clicking on this button connects a crosshair to the mouse. The coordinates of the crosshair are visible to the left of the plot.

j. Not listed is a zoom option which is activated by selecting with the mouse the desired area of the plot that is to be magnified. Notice that since the area is recalculated, zooming requires as much time as the original plot.

`Samples`

The `Sample` button permits the user to save, and later recall, the constellation of values used to produce the current plot. Sample constellations for intersting plots of some models and algorithms are already saved and can be seen by clicking on the `Samples` button. It is also possible to delete samples.

a. `Add sample` Clicking on this button opens a window in which there appears a default name for the sample (the values chosen). The name can be changed as desired.

b. `Remove sample` Clicking on this button opens a window in which are displayed the existing samples for a specific model and specific algorithm. Choose the samples to be eliminated.

2. User input for algorithms

In this section the main categories of user input for the various algorithms are explained. All fields must be completed before the `Start` button actually initiates the chosen routine. For further explanation and a short example of a session download the file *getting started* from the Documentation part of the website www.dss.uniud.it/nonlinear.

Initial values Variables of the model appear in the order they were given in the model definition, see `Model text` in the `Model` window. By assigning intial values the user tells the algorithm at what point in the state space to begin the trajectory. A random choice is not a good choice for the starting point. The user should be either fairly familiar with the system's dynamics or have some guidance (see, for example, *Nonlinear dynamics: computer exercises*, a computer exercise workbook making use of iDMC and available at www.dss.uniud.it/nonlinear).

Parameters Again, parameters appear in the order in which they were given in the model definition. The choice of values is again a delicate matter and the user should have a good idea of range of values that lead to stable asymptotic limit sets, otherwise the trajectory will quickly take on very large values.

Algorithm The input fields required depend also on the type of plot chosen. Other specific terms will be discussed as they appear in particular routines.

a. `transients` The user provides the number of iterations, starting from the des-

ignated initial values, that will not be represented in the graph. If the user wants the entire trajectory plotted the input value is 0. If the user is interested in asymptotic behavior a nonzero input value for transients will be necessary. At just what point the behavior can be considered "asymptotic" is a delicate issue, the choice depends on the model and the objective of the simulation.

b. `iterations` The user provides the number of iterations for the trajectory, beyond the designated number of transients. For example, if the user assigns `transients` 1000 and `iterations` 1500, the algorithm calculates 2500 iterations and plots from iteration 1001 through 2500.

Auto ranges `iterations` The user provides the number of iterations to be used as the basis for calculating the domain and range of the plot. Following the example in b. above, if the user assigns `iterations` 2500 the entire calculation will be performed, the axes calculated and then the points plotted. If, instead, a smaller number is assigned, such as `iterations` 500, the axes are calculated and appear after 500 iterations. In this case another 500 transients are calculated, but do not appear. The final 1500 points are plotted as they are calculated.

Axes The user can decide which variables to plot and on which axes.

3. Absorbing area

This routine allows the user to plot the absorbing area of an iterated map using the method of crtical lines (see, for example, Abraham, Gardini and Mira, *Chaos in Discrete Dynamical Systems*, New York: Springer-Verlag, 1997). There are a number of steps, which should be followed in the order given below. The input fields will be explained in the step descriptions. Notice that the Jacobian matrix must be provided in the model file and currently the maximum is 2 diimensions.

a. `Plot critical set` Pressing this button causes the algorithm to calculate and plot the points for which the determinant of the Jacobian matrix is zero, with precision as indicated by epsilon.

b. `Plot attractor` Pressing this button starts the attractor algorithm which works in the following way. Click on a point of the plot displayed. The algorithm uses the information provided in the fields to plot the attractor initiating a trajectory from that chosen point. To be sure that only the attractor appears, give a sufficient number of transients to be ignored. For a denser plot of the attractor use a higher number of iterations. Not all points on the plot necessarily converge to the attractor and there may be more than one attractor, so that clicking several points is probably useful. Once the plotted attractor is satisfactory, press the `Stop` button to get out of

the attractor plotting mode.

c. `Choose segments` The critical lines method iterates segments which are intersections of the critical sets and the attractor. If steps a. and b. have been followed the intersections should be clearly visible on the screen. Press the `Choose segments` button. Each segment is selected, one segment per dimension, by using 3 clicks of the mouse. First, click near the critical set. Next, click near the left end of the intersection of the set and the attractor. Finally, click near the right end of the intersection of the set and the attractor. If "holes" are present in the chosen segments the approximation of the trapping region will be less acurate (the region will be larger than necessary, but anyway contain the attractor). Press the `Stop` button to get out of segment selection mode.

d. `Clear` Having selected the segments to be iterated, press the clear button. The attractor can be plotted again later, and there is also a button `Hide attractor` that can be used to produce different figures.

e. `Iterate chosen segments` Currently the iteration is accomplished by pressing the button, each time it is pressed another iterate of the segment is plotted, with the given precision. The user can again press `Plot attractor` to see the attractor enclosed in the absorbing area. The `Hide attractor` button permits the user to remove the attractor from the figure, the `Plot attractor` button permits the user to redraw the attractor.

4. Basin of attraction

This routine allows the user to represent the basins of attraction for attracting limit sets over a user-specified region of the state space. In the current version this routine is available only for maps, that is, systems of difference equations. Go to the `File` menu and select from `New plot` the `Basin of attraction` routine.

a. `Basin of infinity` This is the default algorithm which gives a relatively quick idea of the basin of attraction if there is only one attractor. Basically the screen is divided into a grid of initial points which are iterated until they reach the value assigned as infinity or converge to the unique attractor. In this case, points are either part of the basin of infinity or part of the basin of the attractor. The input fields are described in c. below.

b. `All basins` If there are multiple basins this algorithm should be chosen by clicking on `All basins` from the `Plot` menu.

c. `Algorithm` `attractor` The algorithms first iterate a number (designated in the `trials` field) of random initial points and as these trajectories converge to the various attractors, draw points in the limit set on the screen. In order to get all of

the attractors set the number of initial values fairly high, say 100.

The other two input fields improve the appearance and accuracy of the representation. In this routine the number assigned to the `transients` input field is the number of times the algorithm iterates each point on the screen not yet colored as attractors, before checking where the trajectory is. After that number of transients the alorithm checks if the trajectory has hit some part of an attactor appearing on the screen. If this is true, all of the points on the trajectory from the initial grid point through the transients are marked with the appropriate color.

The user can decide to change the rectangle defined in the `Ranges` input field, but the rectangle must include a part of the attractor, otherwise the algorithm finds no visible attractor (and nothing to which points can converge, leaving a black screen). Defining a rectangle which contains only a part of the attractor (without changing the `attractor` inputs) will likely lead to: thinner basins and possibly to the disappearance of some basins if their attractors are not in the selected rectangle; thinner attractors.

Thin attractors may be fattened up by increasing the number of points in the trajectory, that is, the number of `iterations`. Thin basins may be fattened by increasing the number of transients for two possible reasons. If trajectories are slow to converge and the number of transients is low, increasing the number should allow more points of the basin to be plotted. If only a part of the attractor is visible trajectories may be attracted to the part not displayed but if enough transients are allowed (and the attractor is sufficiently "ergodic") the trajectory should eventually reach the visible part of the attractor.

5. Bifurcation

This routine represents the limit sets of a dynamical system as one or two parameters are varied. In the current version the algorithm is available for maps and a rudimentary version is available for 3-dimensional differential systems. An improved routine for differential systems should be available in the next version. The case for discrete-time systems is straight-forward and given below. Differences in applying the routine to systems of differential equations are discussed subsequently. Go to the `File` menu and select from `New plot` the `Bifurcation` routine. In the `Plot` menu the `Fixed initial point` option should be ignored.

a. `Single parameter` The default setting it to plot the asymptotic behavior of the system as a single parameter value is varied over a defined range of values. Under the `Parameters` section the user must select: the parameter to be varied (if there is more than one); the minimum and maximum values to be used for that parameter (`min` and `max` respectively). The `Vertical range` refers to the axis values for the limit sets of the variable which will be plotted on the ordinate axis.

In the `Algorithm` section the user must carefully choose the number of transients to be excluded and the number of iterations to be included in the calculation. In

order to be assured that the trajectory has reached the attractor a sufficiently large number of transients must be excluded, otherwise the plot will contain points that are not really part of the limit set. Further, to avoid misleading plots, the number of iterations should be sufficiently large so as to cover the entire limit set. On the other hand, the total number of transients and iterations is positively related to the time necessary to obtain the plot. The usual trade-off between speed and precision applies. The user must also select which variable to represent on the ordinate axis (if there is more than one).

b. `Double parameter` From the list of parameters two must be selected and the `min` and `max` values chosen for each. This algorithm requires input values for the necessary `Approximation` parameters precision and infinity. Epsilon is an indication at how fine-grained the user wants the plot to be. The smaller epsilon, the closer a value must be to the point to be defined as that point. For example, setting `epsilon`$= 10e-10$ means that all points within approximately 0.0005 radius of a point is taken as that point (and $10e-5 \approx 0.0674$). The `infinity` field defines a cut-off value for unstable orbits. The value assigned to `infinity` tells the algorithm at what variable value the trajectory can be considered on its way to infinity. Very large values will do, but in many cases even relatively small values can be used, unless there are othere attractors existing far from those under consideration. Changes in the values assigned to these approximation parameters have little effect on the calculation speed.

It is the choice of `transients` that does effect the speed but, again, to avoid plotting the limit set and transients approaching it, the value of transients must be sufficiently high. Finally, the plot represents periodic limit sets and the user must fix the highest number of periods (up to 35) to be considered for a given simulation. It will be tempting to take all higher-order cycles as quasiperiodic, but that is not the case.

c. `Bifurcation` for differential systems in 3-d. The bifurcation for continuous systems routine that is currently available works in two steps. First it uses the Poincaré or first-return map to obtain a sequence of points in the plane (see Section 4.4). From this set of discrete points the algorithm calculates the bifurcation map. The `Plot` options are again `Single parameter`, `Double parameter`, `Fixed initial point` and the `Step function` used in the integration routine.

The `Inital values`, `Parameters` and `Vertical range` fields are the same as discussed previously. In the `Poincaré section plane coefficients`field the user must supply the coefficients of the equation that defines the cutting plane. If the dimension is 3 then 4 values are entered. For example if the user enters 1 2 3 4 the resulting plane is $1x + 2y + 3z = 4$.

The `Algorithm` fields must be given some attention. The input field `Transient` regards the sequence of points defined by trajectory intersections with the Poincaré

section (which are not situated at regular time intervals). The user supplies the number of initial (transient) intersections to be ignored. The `Time` field specifies the number of time periods the trajectory and its intersections are observed. The `Step` field is the usual step size for the ODE solution approximation and should be set sufficiently small in order to assure that the intersections with the section are accurately observed.

6. Cycles

This algorithm calculates all k-cycles of a map for a given period and all values of the k periodic points. The algorithm uses an iterative Newton method to solve the system $G^k(x) = (x)$. Once a periodic point is found the others are determined by iteration.

Go to the `File` menu and select from `New plot` the `Cycles` routine. The algorithm is especially sensitive to the user input required to calculate the periodic points. For example, if 2 periodic points are very close and the precision parameter `epsilon` is fairly large, the period-k cycle may not be distinguished because, for that precision, the cycle has a period of $k-1$. The algorithm starts from a random initial point. If the trajectory from that initial value does not converge to a point on the cycle the program randomly chooses the next initial point and so on up to the maximum number of attempts specified in the `max.tries` field. In order to get all cycles this value should be fixed large, depending also on the precision chosen, but even setting 10000 will not reduce the speed too much.

In this version the points are plotted without reference to stability or to which particular cycle the periodic points belong. This should be available in the next version.

7. Lyapunov exponents

a. `Parameter` This is the default routine. The user selects which parameter to vary and gives the `min` and `max` values to be considered. The `Vertical range` refers to the values of the Lyapunov exponents, which typically have fairly large negative exponents and small positive exponents. However most interest centers around zero and positive exponents for which a very small ranges can be used. The Lyapunov exponent is a time-averaged value and sufficient iterations should be specified. If in doubt as to the iterations required for convergence use the `Time` plot over a number of paramter values.

b. `Time` Clear the current plot and select the `Time` option from the `Plot` routine. Click on `Crosshair` from the `Option` menu to see the zero line better and estimate values (crosshair coordinates appear on the left).

c. `Parameter space` This option considers a sub-region of parameter space and

uses a color coding to resperesent the number of positive, negative and zero exponents. The user must supply inputs for the `Algorithm`. The `epsilon` field refers to the Lyapunov exponent. This value gives the symmetric range around zero that is used by the algorithm to define positive, negative and zero and should be chosen carefully. If the point is to distinguish a negative value and zero an epsilon such as 0.001 might be appropriate. It the user wants to make sure that a positive value is definitely positive a less precise value should be specified.

8. Manifolds

The manifold routines are available only for maps, that is, systems of difference equations, for which the manifold of interest is a curve in the plane (see, for example, Nusse and Yorke, *Dynamics: Numerical Explorations*, New York: Springer, 1998). In the current version only manifolds of fixed points are calculated and in some cases the repositioning of the points lying in the lockout region seems to be malfunctioning. Go to the `File` menu and select from `New plot` the `Manifolds` routine.

a. `Unstable Stable Both` From the `Plot` menu the user selects the manifold(s) to be calculated. The default option is `Unstable`.

b. `Right Left Both` From the `Plot` menu the user selects the branch(es) of the indicated manifolds to be calculated. The default option is `Right`. Note that the right and left branches may actually be the same.

c. `Node approximation` The user supplies the approximate position of the fixed point. If the values are close to the fixed point the algorithm will find it.

d. `Algorithm` input fields. `epsilon` specifies how long is the segment starting close to the fixed point. A long segment will extend over more of the manifold at each iterate, consequently, there will be fewer estimated points on the curve. The number of iterates of the given segment is specified in `iterations`.

e. `Lockout region ranges` In the calculation of the manifolds the points on the curve may take on very large values. While the `Plot ranges` defines the screen plot, the lockout ranges define the region in which the calculation occurs. This avoids the problem of overflow errors due to excessively large numbers. Again, in the current version, the repositioning function for points lying in the lockout region may not be working.

9. Shifted and cobweb

These two routines are available only for one-dimensional maps, that is, a single difference equation. Go to the `File` menu and select from `New plot` the `Shifted & Cobweb` routine.

a. `Shifted` The default is the `Shifted plot` which permits the user to plot the values of the variable, shifted forward k periods, on the ordinate axis against the current values on the abscissa. That is, the kth iterate of the map is represented in the (x_n, x_{n+k}) plane. The user provides k in the `Algorithm` field labelled `order`.

b. `Cobweb animation` From the `Plot` menu select `Cobweb animation`. This routine draws the kth iterate of the map in the (x_n, x_{n+k}) plane as in the `Shifted plot`. It also draws the bisector and the forward trajectory using the bisector to reflect back to the abscissa at each pass in order to determine the next iterate value from the curve. The user supplies the `Initial value` and should avoid critical values for the map. The user chooses to view transient behavior by assigning 0 to `transients` or chooses to view asymptotic behavior by assigning `transients` a high value. The motion is slowed down and the speed can be adjusting by dragging on the arrow above the plot. Use the `Stop` button to stop the trajectory plotting.

10. Trajectory

This set of routines form an efficient and flexible way of viewing trajectories and orbits. Go to the `File` menu and select from `New plot` the `Trajectory` routine. Notice that in the `Options` menu for discrete-time models the `Trajectory` plots can be drawn with `Big dots` and/or `Connect dots` to render the results visible to the user. The following description is based on a model represented by a system of difference equations, that is, equation(s) in discrete time. Additional information for systems of differential equations, that is, equations in continuous time, is given subsequently.

Plot There are options for viewing the trajectory.

a. `State space` This is the default choice for systems of more than one variable. The user must simply provide values and then click on `Start`. Other options available: zoom, `Big dots` and `Connect dots` (click `redraw`).

b. `Time plot` The plot represents the time evolution of the variable chosen for the `Range axis`. This is the default plot for one-dimensional maps. To change to `Time plot` click `Reset` first.

c. `Variation` Multiple trajectories can be displayed in a single plot by using this option. The user can increase or decrease the value of a parameter or initial condition by specifying the amount to change at each variation in the second column of input fields that appears when the `Variation` routine is clicked on. Place a 0 in the second field for all parameters or initial conditions that do not change. For example, suppose a parameter parameter `b` is set at 0.2 in the first field and 0.1 is indicated in the second field, while all the other second input fields have zeroes. In the `Algorithm`

part the user must indicate how many simulations should be done. Suppose the user has set this value to 10. The `Variation` routine then produces a plot for which, at each subsequent simulation (after the first), the value of the parameter b is increased by 0.1. The first trajectory is calculated with $b = 0.2$, the second trajectory with $b = 0.3$, and so on.

This option is useful for viewing how limit sets change as a parameter changes, or how sensitive the dynamics are to initial conditions. Simulations are color-coded and the number next to each color defines which simulation is represented by that color.

d. `Automatic bounds` The default for these plots is that the axes ranges are calculated by the algorithm according to the choice made by the user in `Auto ranges` (see Section 2).

e. `Manual bounds` This option allows the user to define the ranges of the axes (in some cases one axis may be determined). This is particularly useful for the `Variation` routine.

Continuous-time systems Differential equations must be integrated rather than simply iterated. The default integrator in iDMC is a Runge–Kutta procedure (`rk8pd`, see description and list of alternatives in appendix 1) which uses the fixed step size provided by the user. To change the integrator click on `Plot` and click on the new choice from the `Step function` options.

The accuracy of the integration is inversely related to the size of the fixed step. The choice of step size depends on the model and on the purpose of the simulation. Setting the step too large may result in such erroneous representations of the curve that the dynamics are not even qualitatively the same. Setting the step very small will, however, increase the time necessary for calculation.

For systems in continuous time the number of iterations and the number of periods are related but not equal. For example, a step size of 0.02 means that the interval between time t and time $t+1$ is divided into 50 equal intervals. The number of `iterates` multiplied by the step size gives the number of time periods. Using 1000 iterations with a step size of 0.02 means 20 time periods have been simulated. Trajectories and orbits are made smoother by fixing smaller steps but the number of iterations must be adjusted accordingly to cover the same time period.

11. Model references

The model directory is composed of well-known prototypical models plus: a directory with models used in computer and analytical exercises (see below) and a directory with a collection of economic models (see below). A reference for each model is given where information on the model, including further bibliographical references, are provided.

Cremona. See Alberich-Carramiana, 2002.
Gingerman. See Phaser software.
Hénon. See Medio and Lines, 2001, chapter 7.
Ikeda. See Alligood, Sauer and Yorke, 1996.
Logistic. See Medio and Lines, 2001, chapter 8.
Lorenz. See Medio and Lines, 2001, chapter 6.
Lv. See Medio and Lines, 2001, chapter 4.
MSmith. See Maynard Smith, 1986.
Nordmark. See Nusse and Yorke, 1998.
Quasi2. See Nusse and Yorke, 1998.
Rossler. See Medio and Lines, 2001, chapter 7.
Rotor. See Nusse and Yorke, 1998.
Silnikov. See Medio, 1992, chapter 4.
Sine circle map. See Medio and Lines, 2001, chapter 8.
Standard. See Alligood, Sauer and Yorke, 1996.
Tent. See Medio and Lines, 2001, chapter 6 and Elaydi, 2000.
Tinkerbell. See Alliggod, Sauer and Yorke, 1996.
Vanderpol. See Medio and Lines, 2001, chapter 8.

Primer models These models are used in connection with the book *Nonlinear dynamics: a primer* (Medio and Lines, 2001) and the accompanying computer exercise workbook *Nonlinear dynamics: computer exercises* available at the software website www.dss.uniud.it/nonlinear.

Con2d. See Medio and Lines, 2001, chapter 2.
Cona. Model created by Gianluca Gazzola as exercise.
Conb. Model created by Gianluca Gazzola as exercise.
Conlyapa. See Medio and Lines, 2001, ex. 3.11 (b).
Conlyapb. See Medio and Lines, 2001, ex. 3.11 (d).
Conbif. See Medio and Lines, 2001, ex. 5.6.
Conlocal. See Medio and Lines, 2001, ex. 3.11 (a).
Conpar. See Medio and Lines, 2001, chapter 1.
Disa. Model created by Gianluca Gazzola as exercise.
Disbif. See Medio and Lines, 2001, ex. 5.8 (c).
Dispar. See Medio and Lines, 2001, chapter 1.
Disparlag. See Medio and Lines, 2001, chapter 2.
Flip. See Medio and Lines, 2001, ex. 5.8 (a).
Hopf. See Medio and Lines, 2001, ex. 5.7.
Quasi. See Medio and Lines, 2001, chapter 4.
Vanderpol. See Medio and Lines, 2001, chapter 8.

Economic models These are models used in teaching with the book *Advanced Macroeconomics* (Romer, 2001) and the accompanying computer exercises *Macroeconomic models: computer exercises* available at www.dss.uniud.it/nonlinear.

BH. See Brock and Hommes, 1989.

Cournot. See Puu, 2003.

Cournotad. See Puu, 2003.

Diamond. An overlapping generations model with logaritmic utility and Cobb-Douglas production. See Romer, 2002 chapter 2.

Olg1. See Medio, 1992, chapter 12.

Olgns. See Medio and Lines, 2001, ex. 5.11 and Medio, 1992, chapter 12.

Ramsey. A macroeconomic model with logaritmic utility, Cobb-Douglas production and technology. See Romer, 2002 chapter 2.

Solow. The Solow growth model in per-capita terms. See Romer, 2002 chapter 1.

Alberich-Carramiñana, M. 2002. *Geometry of the plane Cremona maps*, Berlin: Springer-Verlag.

Alligood, K. T., Sauer, T. D. and Yorke, J. A. 1996. *Chaos: and introduction to dynamical systems.* New York: Springer-Verlag

Brock W. and Hommes,C. 1989. Heterogeneous beliefs and routes to chaos in a simple asset pricing model *Journal of Economic Dynamics and Control* **22**, 1235-1274.

Elaydi S. N. 2000. *Discrete chaos*, Boca Raton, FL: Chapman & Hall/CRC Press.

Maynard Smith, J. Evolution, games and learning *Physica D* **22**, 43-49.

Medio, A. 1992. *Chaotic dynamics. Theory and applications to economics.* Cambridge: Cambridge University Press.

Medio A. and Lines M. 2001. *Nonlinear dynamics: a primer.* Cambridge: Cambridge University Press.

Nusse H. E. and Yorke J. A. 1998. *Dynamics: numerical explorations.* New York: Springer-Verlag

Phaser software, see site at www.phaser.com

Puu, T. *Attractors, Bifurcations, & Chaos - Nonlinear Phenomena in Economics*, Berlin: Springer-Verlag, 2003.

Romer D. 2001. *Advanced Macroeconomics.* New York: McGraw-Hill.

12. New models

New models can be introduced without having to re-compile iDMC because they are written and read by the powerful **Lua** programming language (see below and the site www.lua.org). The easiest method is to use an editor to call up an existing model, make the necessary changes and save the new file, for example, mymodel.lua, in the directory *Models*. The basic structure for the model file can be seen in the logistic model example below:

```
name = "Logistic"
description = "See Model Info"
type = "D"
parameters = "mu"
variables = "x"
function f(mu, x)
y = mu*x*(1 - x)
return y
end
function Jf(mu, x)
return mu - 2 * mu * x
end
```

For continuous-time systems change `type` to "C". Typical mistakes are forgetting to put the parameters and variables always in the same order when defining functions and not making proper use of quotation marks.

The Lua language (Copyright, 2003 Tecgraf, PUC-Rio) is "a powerful, light-weight programming language designed for extending applications". This description and the following excerpts are from the *Lua 5.0 Reference Manual*, Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes, Tecgraf Computer Science Department PUC-Rio, 2003 (downloadable from www.lua.org). That document describes version 5.0 of the Lua programming language and the Application Program Interface that allows interaction between Lua programs and their host C programs. For our purposes only the following characteristics of the language are central.

Page 3. Numerical constants may be written with an optional decimal part and an optional decimal exponent. Examples of valid numerical constants are

3 3.0 3.1416 314.16e-2 0.31416E1

Pages 8-10. Lua supports the usual arithmetic operators: the binary + (addition), - (subtraction), * (multiplication), / (division), and ^ (exponentiation); and unary - (negation).

The relational operators in Lua are

`== ~= < > <= >=`

These operators always result in `false` or `true`. Equality (==) first compares the type of its operands. If the types are different, then the result is `false`. Otherwise, the values of the operands are compared. Numbers and strings are compared in the usual way. The operator = is exactly the negation of equality (==). The order operators work as follows. If both arguments are numbers, then they are compared as such. Otherwise, if both arguments are strings, then their values are compared according to the current locale.

The logical operators in Lua are

`and or not`

All logical operators consider both `false` and `nil` as false and anything else as true. The operator `not` always return `false` or `true`. The conjunction operator `and` returns its first argument if this value is `false` or `nil`; otherwise, `and` returns its second argument. The disjunction operator `or` returns its first argument if this value is different from `nil` and `false`; otherwise, `or` returns its second argument. Both `and` and `or` use short-cut evaluation, that is, the second operand is evaluated only if necessary.

Operator precedence in Lua follows, from lower to higher priority:

or
and
< > <= >= ~= ==
..
+ -
* /
not - (unary)
^

You can use parentheses to change the precedences in an expression. The exponentiation operator is right associative. All other binary operators are left associative.

Page 50. The library is an interface to most of the functions of the standard C math library. (Some have slightly different names.) It provides all its functions inside the table math. In addition, it registers the `global_pow` for the binary exponentiation operator ^ so that x^y returns x^y. The library provides the following functions:

`math.abs math.acos math.asin math.atan math.atan2`
`math.ceil math.cos math.deg math.exp math.floor`
`math.log math.log10 math.max math.min math.mod`
`math.pow math.rad math.sin math.sqrt math.tan`
`math.frexp math.ldexp math.random math.randomseed`
`math.pi`.

Most of them are only interfaces to the corresponding functions in the C library. All trigonometric functions work in radians (previous versions of Lua used degrees). The functions `math.deg` and `math.rad` convert between radians and degrees. The function

`math.max` returns the maximum value of its numeric arguments. Similarly, `math.min` computes the minimum. Both can be used with 1, 2, or more arguments. The functions `math.random` and `math.randomseed` are interfaces to the simple random generator functions `rand` and `srand` that are provided by ANSI C. (No guarantees can be given for their statistical properties.) When called without arguments, `math.random` returns a pseudo-random real number in the range [0, 1). When called with a number n, `math.random` returns a pseudorandom integer in the range [1, n]. When called with two arguments, l and u, `math.random` returns a pseudo-random integer in the range [l, u]. The `math.randomseed` function sets a seed for the pseudo-random generator: Equal seeds produce equal sequences of numbers.

For the following formats refer to the indicated model file which uses that format: if then, see cournot.lua; math.mod, see sinecircle.lua; for inserting many constants and the use of intermediate variables, see quasi2.lua.