DANISH **R**ESEARCH **U**NIT FOR **I**NDUSTRIAL **D**YNAMICS

DRUID Working Paper No. 98-5

Laboratory for Simulation Development

by
Marco Valente
February 1998

Laboratory for Simulation Development

Ver. 1.0

22nd January 1998

by Marco Valente

Abstract

This document contains a description of the main purposes and characteristics of Lsd. It discusses the problems related to the use of simulations in social sciences and describes how Lsd tackles these problems.

In general, the use of simulations is constrained by two kinds of problems: difficulties in building the programs and difficulties to make use of other people's programs. Lsd proposes a system to facilitate both aspects of the use of simulation models. This document is mainly concerned with the use of simulations by unskilled computer users, and describes in detail how Lsd can be successfully used to explore and use a model without requiring any programming knowledge. Though it is not the main concern of the document, there are also few hints on the aspect of simulation program building.

Keywords

Programming, Simulation models.

JEL Classification

C63

ISBN 87-7873-041-4

Contents

Introduction

Lsd is a system developed to facilitate writing, use and presentation of simulation models in social sciences. The use of simulations is becoming a widespread tool in these sciences, both because of the cheap computing power available and of the difficulties in using alternative tools for the problems in these fields. There are, anyway, many drawbacks in the use of simulations, stemming mainly from the fact that they involve the use of computer programs by an audience that is not normally trained in computer science. We can separate two different sources of problems, for writing a model and for using other people's model.

Writing a Simulation Model

Writing programs for simulation models entails many technical details not directly related to the model itself, but necessary for its implementation. Though these technical details are basically similar for any model, it is often necessary to re-write the whole set of "service" code for any new model, so that modelers have to spend much of their time to write code not related to the model, but yet crucial for its results. This technical code necessitates specific knowledge on file systems, interfaces, memory management and, in general, all the technical aspects of a program. Lsd provides the possibility of writing a simulation model by writing exclusively the code for the equations of the model, while all the "technical" details are automatically arranged by the system. Yet, Lsd does not constrain modelers to implement only specific classes of models: the run-time version of a Lsd model is nothing but a C++ program, thus it is possible to implement virtually any computational structure. Lsd provides a set of utilities that allow modelers to write only the code for the model and to extremely simplify the most common operations.

This document does not discuss in detail the utilities for models writing, though it sketches the general philosophy used. Rather, it presents the set of interfaces (introduced in the next paragraph) final users have available to use an existing model. It needs just to remark that these interfaces are automatically generated when loading the model and do not require any specific coding by the model writers.

Presenting Simulation Results

The second type of problems in the application of simulation models in social sciences is due to the fact that even relatively simple simulation models are far too long to be described in detail in a scientific paper. Authors rightly prefer to discuss the contents and results of the model, rather than their technical implementation, so that simulation results are generally accepted with some skepticism on the way they were obtained. The best solution should be to distribute the simulation

program along with the article, so that referees and interested readers could see the implementation of the model, test the simulation runs presented in the paper and possibly try different settings, to test the robustness of the claimed results. This is normally not done for many reasons. Using someone else's code is never a simple task. The code modelers in social science generally manage to write is not endowed of user-friendly interfaces and not well documented. Often crucial choices on parameters' values are embedded in the code and modifiable only by digging in the source code, making difficult even only finding the line with parameters assignment[1]. Readers are generally not trained in computer science, so the task of using the authors' code, even when available, can be overwhelmingly difficult.

Lsd provides users of simulation models with ready-to-use executable programs, which allow a full understanding of the models content and the possibility to re-run the original simulations (or testing different parameter settings) without requiring any computer knowledge other than the one necessary for commonly used computer programs. In particular, any computer unskilled computer user can perform the following actions by simply using the mouse:

- Browse through the structure of the model, to have a global understanding of the model contents;
- Check the equations' code, to see the actual implementation of the computational part of the model (authors can include any comment to their equations and, anyway, most of the commands are self-explaining even to naive computer users);
- Checking all the parameters and, in general, initial values proposed by the author of the model;
- Make their own simulation runs;
- Observe the results, both via run-time plotting of the most important variables and by post-simulation plotting of any variable of the model;
- Test any combination of parameters, and in general initial values, and run the relative simulations;
- Run automatically batteries of simulation runs with different random number seeds to test the robustness of the results.

A model written in Lsd is equivalent of a program written in C++, one of the most powerful, fast and commonly used languages currently available. It means that the only limitations to the dimensions and speed of the models are given by the systems the model is run on.

This document is meant to provide a general understanding of Lsd models, from the viewpoint of users of an existing model. Thus, it does not discuss issues related exclusively to the tasks of building models (e.g. the debugging facilities in Lsd), contained in the Programmers Manual for

[1] There are many good programs written by social scientists, available to the public and easy to use. This paragraph does not, of course, refer to any particular simulation model, other than the ones produced by the author himself.

Lsd. The first section contains a description of the general structure of models developed in Lsd. The other sections present the use of the interfaces in Lsd. As example during the presentation is used the Lsd implementation of a model originally presented by Nelson and Winter in 1982. The first section contains also a brief illustration of this model, to better follow the example.

A Lsd Model

Objects

A model in Lsd is based on the Object Oriented approach. That is, its structure is made of a set of Objects. Objects are abstract entities, referred to with a name, or label, which can contain either other Objects, or numerical variables. Objects are devoted to represent entities of reality the model simulates, like, for example, Markets, Firms etc. An Object is assigned a label so to distinguish it from other Objects. Once an Object is defined, it is possible to implement it in any number of copies in the model. In the following, we will refer to an Object type, indicating the nature of the Object, and to the instances of the Object, to indicate its actual copies in the program.

Variables and Parameters

The behavior and status of an Object are determined by the elements it contains. They can be of two different types: Variables (which will be referred to by using the initial capital to distinguish them from generic variables) and Parameters. Both these elements are nothing but numerical values associated to labels. The difference is that Variables have a piece of code associated to them, while Parameters don't. The piece of code associated to a Variable can do any computation, but normally it computes a numerical value for the associated Variable. Parameters, instead, remain constant through all the simulation run, unless the code associated with some Variable modifies their value. In the following we will discuss about lagged values for Variables. In fact, the numerical values of Variables are time tagged, since the general structure for the equations is that of difference equations. The model uses as many lagged values for Variables as necessary in the equations (a model whose equations don't contain any lagged Variable is either trivial or inconsistent). The users will then need to provide as many initial values for as many lagged have been defined in the Variable. Of course, Parameters cannot have any lagged value.

Structure of the Model

An Object can contain other Objects. Depending on the model, it is possible to either include one type of Object within another Object or to keep the two independent. Even in cases where two Objects are perfectly identical (that is, same Parameters, Variables and equations), the two cases represent different choices made by the modeler. In case one Object is contained in another, the instances of this Object will automatically refer to the instance of their parent Object that contains them. Instead, in case the two Objects types are defined as independent one another, their instances don't refer automatically to instances of the other type.

For example, consider a model with two Object types: Firms and Markets. If you place Firms as Objects contained in Market, it means that any time a Firm needs to refer to a Market (e.g. to know the current price or to sell its production), it will automatically refer to one specific instance of Market (the one containing it). Instead, in case the two Objects are defined as independent, any instance of Firm will need a specific routine to decide to which of the (potentially many) instances of Market it has to refer to. Hence, the structure of a model is relevant in respect of the results obtained.

Simulation Run

A simulation run consists in computing repeatedly all the equations in all the instances of every Object in the model. We will refer to each repetition as "time step". Each equation is computed once and only once for each time step[2]. When an equation is computed, its associated piece of code is executed. The result of such computation is a numerical value that is assigned to that Variable for that time step.

Users can decide which Variables' values to observe during the simulation run, and automatically Lsd will plot the graph of this values while the simulation is running. Analogously, users can determine a set of Variables to save[3]. Lsd will create result files containing all the Values for every Variable indicated. After the simulation, it is possible to use the Result Analysis module to plot all or parts of the values contained in the result files.

It is possible to automatically run a battery of simulation runs. For example, 50 simulation runs, with the same model and initial values, but different pseudo-random numbers. In these cases, besides the result file for each simulation run, Lsd creates also a summary file containing all the

[2] This is the default choice in Lsd, but authors can always allow an equation to be computed as many times as desired. As we will see below, the normal Lsd philosophy is based on providing default mechanisms that apply to the most common cases, but models' authors can always override these default mechanisms.

[3] Each parameter setting can be separately saved and the result files will be named after the name given to the model setting.

final values for each simulation run. All result files are standard tab-delimited text files that can be loaded in statistical programs to perform further analysis.

The time of execution of simulation runs depends, of course, on the type of model and the computer used. But models in Lsd are comparably very fast, because they are in essence pure C++ compiled code. Moreover, users have different options that allow to decide the amount of information provided by the system during the simulation run, so that it is possible to minimize this information and run the model at maximum speed.

Notes on Modelers' Requirements

Though this document is not meant to illustrate how to write a model in Lsd, this paragraph discusses the required tools and minimum knowledge a modeler is requested to use Lsd for writing a model.

The structure of the model (that is, the Objects, their reciprocal relations, the Variables and Parameters contained) is defined by simply using graphical interfaces, and in general takes no more than few minutes to implement it[4]. The most difficult task is the definition of the equations for the model. For that it is necessary to use a C++ compiler (Lsd used Borland C++, ver. 4.5 and superior for Windows and GNU gcc compiler for Linux and Sun Solaris), and so modelers are requested to know how to compile a program[5].

The programming capabilities requested depend, of course, on the complexity of the model. But they are extremely limited in respect of the knowledge necessary to write a dedicated program for a model In fact, the equations are written independently one another as if they were difference equations. That is, model writers do not need to decide the actual order of execution of each Variable, since this is decided at run-time by the system, but only to write each equation considering that it has to produce the result for that Variable at any generic time step. Moreover, every information about the model (that is, the values of other Variables or Parameter in the model) is indicated in the equations only by using their labels. In case many variables of the same type, modelers can still avoid to specify the exact instance of the necessary variable because the Object Oriented structure of the model resolves the problem of choosing the one that needs to be used in any computation.

[4] Provided that model writer knows already the structure to implement, of course. Anyway, it is always possible to edit the model structure at any stage of the model implementation.

[5] The Lsd distribution for models' writers has also makefiles so that the actual problem is limited to have a compiler. The installation and the Programmers' Manual describes how to accomplish the compilation.

In summary, writing a model with Lsd is mainly a task of writing the equations for it, in an extremely simplified format based on C++, and augmented with the Lsd library of functions. Lsd will use this information to automatically prepare all the interfaces and to run the model as described in this document.
Since the equations relate to the rest of the model only via Variables' labels, they can also be easily exported in different models, provided that in the new model exists somewhere (even in completely different Objects) Variables with the same labels as the ones used in the equation. The same applies also for Objects or sets of Objects. This implies that a model can be implemented gradually, by beginning with an extremely simplified version and then adding more and more components or revising existing ones, without the risk of disrupting the work previously done.

All the interfaces described in the next sections are automatically added to the model, without need for work by the modelers, besides the equation code. When a model is finished, it can be distributed as a stand-alone program[6] that can be run either under Windows or under Unix (exactly the same code can be implemented for the two platforms). The distribution of Lsd for modelers includes also project files (for Borland IDE) and makefiles (for GNU gcc) that allow to compile the model in a very simplified way.

The Example Model: Nelson and Winter (1982), ch.12

The model used as example in the following sections is an implementation of the model by R.Nelson and S.Winter, presented in 1982 in their book Evolutionary Theory of Economic Change, chapter 12. It refers to a market where a homogeneous product is traded. A number of firms are allowed to make investments and research. Investments provide higher levels of capital and hence of productive capacity. Research provides higher productivity to yield higher volumes of production for a given amount of productive capacity. Firm exploits fully their capacity, producing the maximum amount of production given their capital and productivity. The price is obtained as a function of the total production of the market. Research can be done either by imitation of more advanced competitors or by innovation, that is searching for brand new productive methods. The results of research depend on the dimensions of the firm and on a random function.
Though simple, the model entails a rather high number of initial values and parameters, and, being one of the most famous simulation models in Evolutionary Economics it is a very good basis for developing new models, as modification of this base version.

[6] First time users of a Lsd model need to install a set of libraries for the graphical interfaces. The package providing these libraries (Tcl/Tk) is freely available as a self-installing program.

The variables' labels used in the model are the ones used in the original presentation of the model. Their meaning is reported below:

Q_TOT: Total quantity of production on the market.

P: Price

A_MAX: Maximum productivity currently used among the firms

PROF: Profits per unit of capital

K: Capital

RIM: Proportion of expenses for imitation research per unit of capital

RIN: Proportion of expenses for innovation research per unit of capital

A: Productivity

Inn: Parameter indicating whether the firms makes research both by innovation and imitation (Inn=1) or only by imitation (Inn=0)

Q: Quantity produced in each firm

Id: An identification value, different for every firm

A_IM: Productivity obtained via research in imitation

A_IN: Productivity obtained via research in innovation.

The rest of the document assumes the reader has available a copy of the Lsd implementation of the Nelson and Winter model (downloadable at http://www.business.auc.dk/~mv/lsd_home.html). The document contains anyway pictures of the main windows of the program, so that it is possible to evaluate Lsd even without the actual software available.

Load and Browse a Model Structure

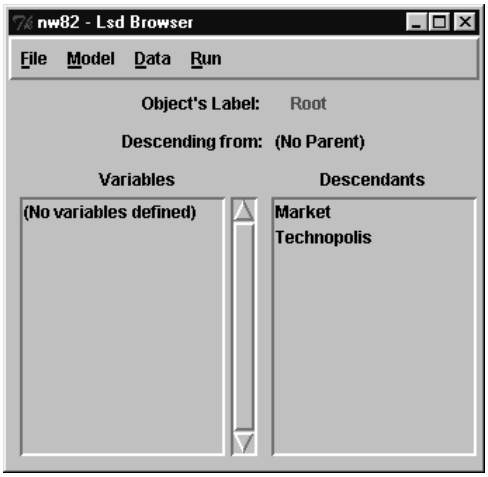
There are two windows appearing at start time. The one labeled "Log" serves to report messages from the system and for managing the options during the simulation run, and hence it will be discussed later. The other window allows to browse through the model structure and to observe the contents of the Objects. This window will be referred to as Browser, and it shows the content of one single Object type per time. The header of this window shows a default name for a model, Sim1, assigned when no model has been specified. If no model is loaded, like at start time, it shows the only Object present in every model, named Root, which does not have any variable or descending Object.

A program in Lsd is partly embedded within the program and partly it is contained in data files. The part within the program is composed by the equations, which cannot be modified by end-users of a model. The data files contain the structure definition, the numerical values necessary to run a simulation and the settings for a simulation run (e.g. number of steps). These files can be loaded, modified and saved also by end-users of the model, so that it is possible to keep track of the parameter settings associated with the results.

In order to load a model use the entry Load in menu File[7] (the data files have the standard extension "lsd'). Loading the file "nw82.lsd", the window will appear as shown Figure 1. The model name is changed showing the name of the model loaded. The window shows again the first Object in the model hierarchy (that is, "Root") but now it contains two Objects: Market and Technopolis.

Immediately under the menu bar, the window reports the Object label the Browser is showing and the name of the Object it descends from. The two boxes in the bottom contain, as indicated by the titles, the set of variables (that is, both Variables and Parameters) of current Object and the set of descending Objects. The Root Object does not contain any variable, and shows the labels of the two Objects above mentioned.

Figure 1 – Model Browser on Root

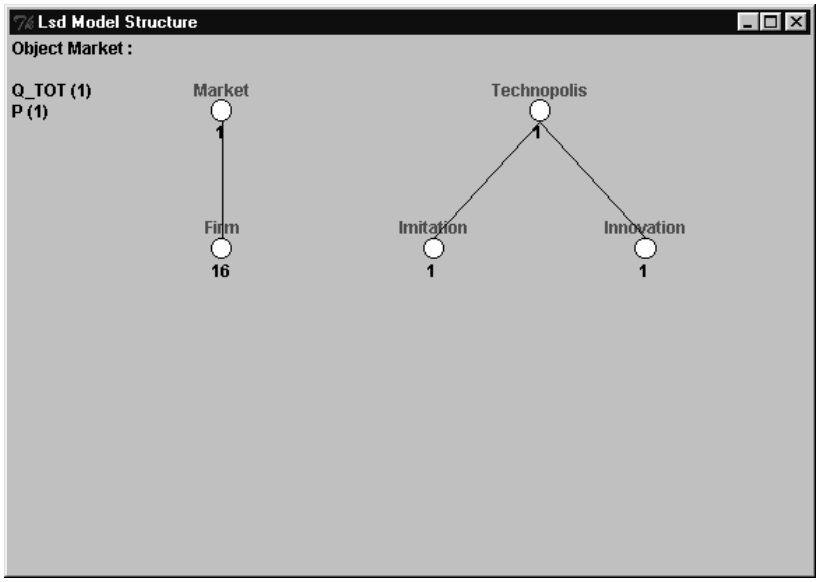


To move the Browser so to explore another Object, you need to follow the structure of the model. That is, the Browser can be moved so to show the content of either one of its descending Objects or the parent Object. Double-click on the name of the descending Object to move "down" the structure and click on the name of the parent Object to move "up".

[7] The menus can be activated, besides clicking on them, by typing the underlined letter while keeping the key Alt pressed. Pressing F10 also activates the menu bar, so that it is possible to use the arrows to move through the entries. Many entries inside the menus can be activated using shortcuts. These are indicated on the side of the entries in the form "Control+X", meaning that the key x has to be used while keeping the key Control pressed.

There is also a new window opened after loading a model, named “Lsd Model Structure”. It contains a graphical representation of the model, and allows also give some commands to the Browser. For the model “nw82” it appears as shown in Figure 2.

Figure 2 – Graphical Representation



This window presents a graphical representation of the model’s structure. That is, Objects’ names number of their instances, their content and relations[8]. Each Object is indicated by its name, in red characters, and by a small graphical symbol; the numbers below the Objects represent the number of instances for each Object type. In the above example, all the Objects are present with only one instance, but Objects Firms having 8 instances.

The lines indicate the hierarchical relation among Objects: the higher Objects contain the lower Objects in the picture. Thus, for example, Object Firms are contained in Object Market, and both Object Imitation and Innovation are contained in the Object Technolopolis. But Market and Technopolis are independent. Moving the mouse pointer over one of the Objects causes the appearance of some writings on the left side of the window. They indicate the name of the Object and the list of the variables contained. For example, passing over the Object Market (as in the case of Figure 2), the window will tell that there are two variables, Q_TOT, A_MAX and P (the meaning of the characters between parenthesis will be explained later).

The graphical window provides a global vision of the model structure. For a more detailed

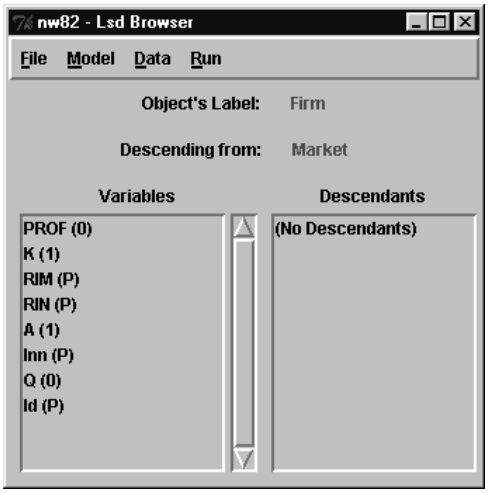
[8] Only the Object Root is not represented, both because it is necessarily present in any Lsd model and, normally, it does not contain any data relevant for the model.

observation of the model it is necessary to move the Browser window to point on the Object of interest. Besides the way explained above, the Browser could be moved directly to the desired Object from the graphical window by double-clicking on its symbol.

Equations in Lsd

In the previous paragraph we have seen how to have a general understanding of the model structure. In order to observe the Variable's equation you need to point the Browser on the Object containing the Variable you want to observe. For example, move the Browser on the Firm Object, either by double-clicking on its symbol on the graphical representation of the model, or by clicking on the label Market and then Firm in the Object box of the Browser. When you have done this, the Browser window will appear as in Figure 3.

Figure 3 – Browser on Object Firm

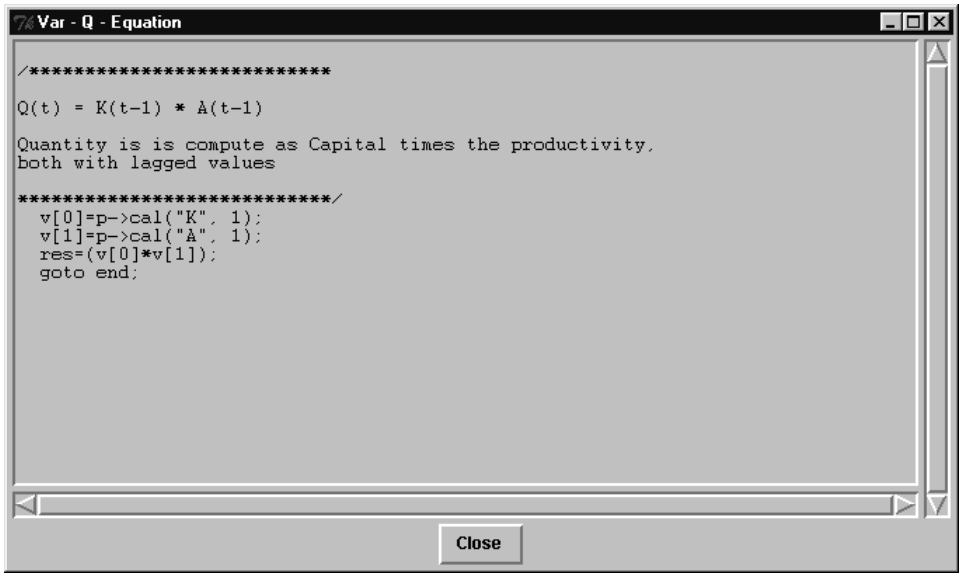


The list of variables reports several labels followed by either a number or P. The labels followed by P are Parameters, while the numbers are associated only to Variables and indicate how many lagged values is has to store for the models' equation. For example, Variable K has, at any time steps, two values: one is the present value and the second one refers to the previous time step. Variable PROF, instead, does not store any lagged value, since they are not necessary for the equations of the model. It means that the equations of the model make use of the one period lagged value of Variable K (precisely, the equations for K itself and for Q use the lagged values), while Variable PROF is always used only with its current value.

As said in the introduction, the equations for a model are written in Lsd as C++ code. Actually, it is an extremely simplified code that makes use of a set functions that, together with standard C++, allows to write the equation in a very simple way. For example, consider the equation for Variable Q, that is the quantity produced by each Firm.

In order to show the equation of a Variable you need to double-click on the desired Variable. A small window appears, containing a set of options we will discuss later (this window is shown in Figure 9). Press the button Equation in this window and a new window like the one shown in Figure 4 will be created. A shortcut to show the Variable's equation is to click with the right button of the mouse on the name of the desired Variable.

Figure 4 – Equation Code for Variable "Q"



The window reports exactly the piece of code used to compute the equation of the Variable. Even though it is C++ code, the structure of the equation should be easily understood. Moreover, as in this case, model writer can include comments in the equation's code.

The code reported in the window is the equivalent of the equation

$Q_t = K_{t-1} * A_{t-1}$

The first two lines in the equation use the Lsd function "cal" to ask the system to provide the values of K (one period lagged, as indicated in the second field of "cal") and of A. The equation uses two local variables, v[0] and v[1] for sake of clarity, to which assigns temporary the values of the two lagged Variables K and A. The product between these two values is assigned to the variable "res". Such variable is used by the system to assign the result of the equation to its Variable, so any

equation in Lsd terminates with an assignment to "res". The last line is just a technical directive saying that the equation code for this Variable is finished.

It is possible to double-click on the name of one Variable in the window for the equation to produce a new equation window for the clicked Variable, so that it is possible to follow the chain of computations in the model.

Technical Notes on Equations' Code

There are two important things to note about the code above presented. And specifically, two things that are missing from the above code and are normally present in simulation models. First, since there are many Objects Firms, there are also many instances of Variables named "K" and "A", that is, one for each instance of Firm in the model. The model writer does not need to worry in the equation about how to find the correct instance, since Objects' structure allows the identification of the correct instance. In fact, by default, the equation for each Variable will use the Variables present in the same Object, or in one of its immediate relatives[9]. The programmer, therefore, can directly use the name of the necessary Variables in the equations, without using indexes, addresses, or any other means for tracking variables normally used in programs.

The second important (missing) fact is that the equation does not contain any control on the state of the Variables A or K. That is, it should be possible that, given the sequence used for the computations of Variables in a time step, the equation for Q is computed when A and K have not been yet updated. Hence, their one-period lagged values are actually two-period lags. The model writers do not need worry for this either, because Lsd takes care of checking the time of update for the requested Variables so that it always reports the correct values.

The idea is to allow the programmer to write the equations of a model as he could write them in mathematical terms on paper as difference equations, using an intuitive default system to complete the information necessary to perform the task requested. Several Lsd functions (like summarization or sorting functions) are available to facilitate the most common operations.

Hence, modelers can rely on the default systems to write as little code as possible. But it is always possible to override the default in case of particular needs. In fact, the code for the equation, as well as the whole code for Lsd itself, is written in C++, and hence does not have any constraints on the possibilities of computational expressions. In theory, it should also possible write the whole simulation program in one single "equation", without using at all the Lsd facilities.

[9] Precisely, the Variables are searched first in the same Object of the Variable under computation; then in the descending Objects; and then in the parent Object. The searching procedure is recursive, so that it follows the same strategy in each of Objects encountered, ensuring that the whole model, if necessary, is explored.

Note also that representing the equations as pure C++ allows also to easily plug external code in a simulation model, so to make use of external modules.

Initial Values

We have seen how to observe the structure of a model and the Variables' equations. There is one relevant aspect left in order to understand in detail the content of a model, that is the initial values. There are three different types of initial values necessary for a simulation run of a model:

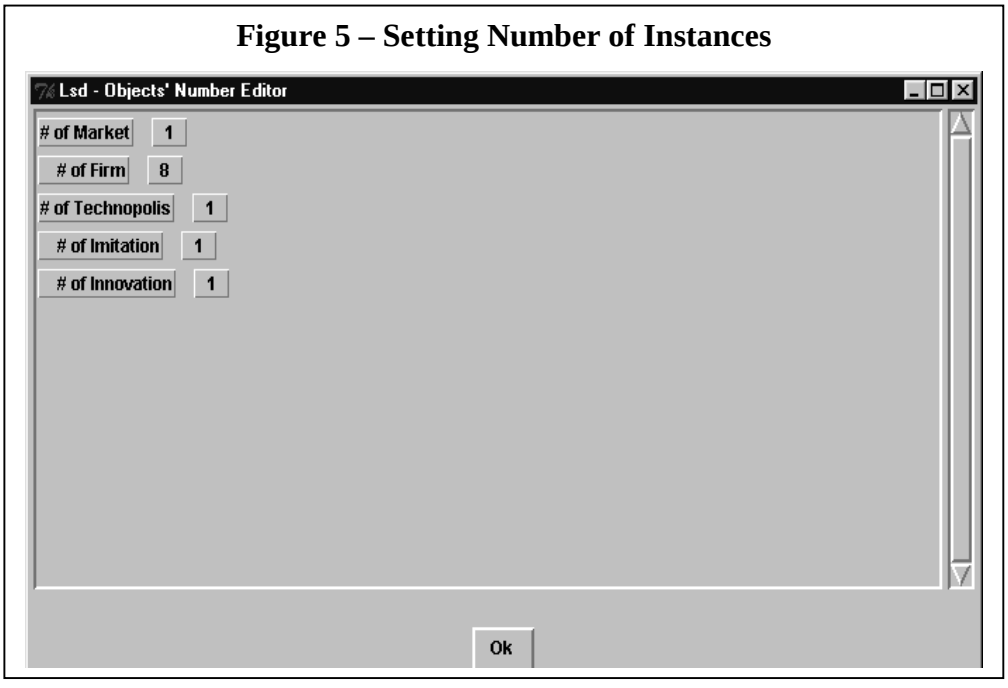
- Number of instances for the different Objects
- Values for Parameters
- Lagged values for Variables.

Note that the number of each type of Object determines also the number of Parameters and lagged values for Variables to initialize, since these are contained in the Objects.

Set Number of Instances

To open the interface to modify the number of instances of the Objects in a model you need activate the entry "Object Number" in menu "Data". The window will become as shown in Figure 5.

Figure 5 – Setting Number of Instances



This window presents the whole set of Objects in the model by using their labels, followed by the number of instances currently present in the model. The indentation indicates the hierarchical relation among the Objects, so that Firms is shifted on the right in respect of the label Market above,

since Firm is a descendent from Market. Technolopolis instead is again starting from the left margin, as the Object Market.

To modify the number of Objects click on the number of the Object you want to change, for example the number 8 along the label Firms. The system will ask you the new number of Objects.

There are two different behaviors resulting from changing the number of instances for an Object type: they depend on whether you have increased or decreased the former value. In case you increased the number the window will return as the previous one, showing, of course, the new number of Firms[10]. The new Objects created in this way are all identical to the very first of the sequence. That is, the initial values (Parameters' value and Variables' lagged values) are copied from the very first Object of that type in the model. These values are used as default, but Lsd does not permit to use them for a simulation run. If you now tried to launch directly a simulation run, the system will not do that and will issue a warning. The user needs, at least, to open the editor to confirm that those values are accepted. The next paragraph will say how it is possible to edit these values.

In case you decreased the number of Objects, a new window will appear asking which instances you want to eliminate. Two options are available; a fast one, allows to eliminate the last Objects in the series. The second option allows you to choose which instances you want to eliminate, by indicating the ordinal numbers in the set of Objects to be reduced. You can try both options. Consider that the values are not saved in a file unless you explicitly do that or you launch a simulation run.

[10] The graphical representation of the model will update the values reported only when you exit from this window and you return to the Browser

Figure 6 – Multiple Groups

The example we have seen till now have a very simple structure. Consider instead a case where the model contains more Objects Market. In this case there will be one group Firm descending from each instance of Market. It is possible to have the number of Objects descending from the different instances of parents to different values. To test it, try to increase the number of Markets from 1 to, say, 3. The new Objects number list will appear as in Figure 6.

As you can see, each new Object Market has been created by default with one descending Firm. There are also three lines for Object type Firm, each referring to a specific Market. In the initial setting, since there was only one Market, there was no need to specify which Market the set of Firms was referring to. Now Lsd created the index for different Markets so to differentiate the different groups of Firms.

Now, changing the number of Firms in the model you can choose whether the new number has to be applied to all the sets of descending Firms or only to the one you clicked on. Try to click on one number on the side of one of the label Firm. After having inserted a value, check the box along the entry “All Equal”. After having pressed the Ok button, you will see that all the groups of Firms in the model have been set to the new value. Note that, when using the “All Equal” option, is not possible to choose the instances to delete, in case the new value is lower for some group. The last Objects will be deleted automatically.

The creation of the indexes to differentiate among different sets of descendants is one important characteristics of Lsd, because it is the only way to differentiate among entities otherwise identical.

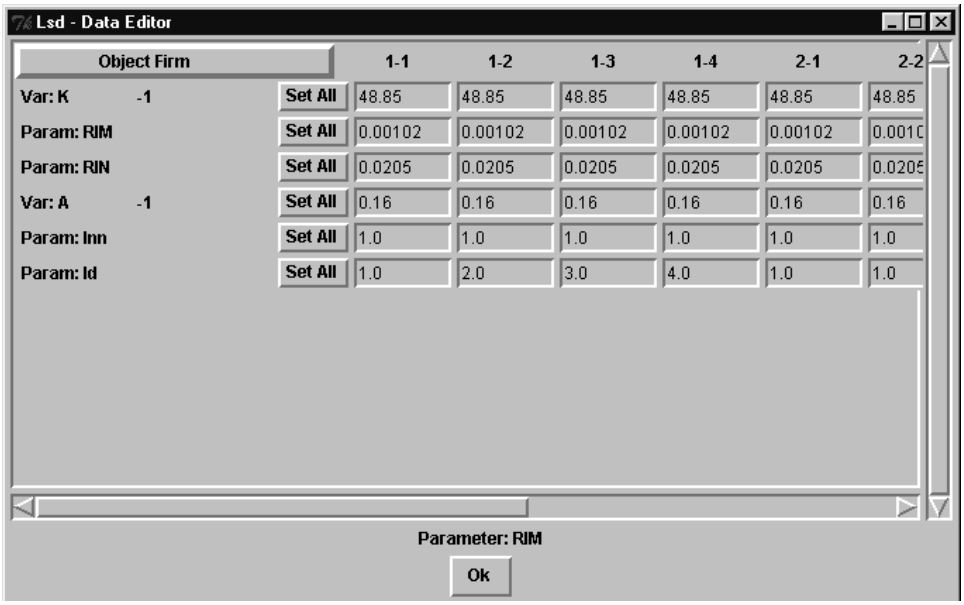
The indexes can also become rather complex. In particular, consider a model with a “deeper” hierarchy. For example the same model as here, but the Objects Firms containing descending Objects, say Plant. In this case, Lsd would create a two-digit index to distinguish the different instances of Plant: the first digit would refer to the higher level Objects (e.g. Market) and the second to the lower level (e.g. Firms). We will see later how this index is used, even in the case of the present structure of the model, in order to differentiate among the different instances of the Variables in the model.

Setting Initial Values for Parameters and Lagged Variables

Once the number of Objects has been determined, the user can observe and edit the initial Values for Parameters and Variables. It is possible to edit the initial values for only one Object type at each time. For the following example, set the number of Objects Market to 3, each with 4 Firms. To exit the window and return to the Browser click on the Ok button.

Move the Browser to point on Object Firms and then activate the entry “Init. Values” in menu “Data”. The window will become as in Figure 7.

Figure 7 – Initial Values for Variables



The system reads the structure of the model and prepares a spreadsheet-like window showing the current values for each Variable or Parameter in the Object indicated in the label on the upper left corner. Note that not all the variables of the Firm are actually listed in this window. In fact, the system includes only the elements that actually need an initial values, that is all the Parameters and

lagged values for Variables. The line headers indicate to which Variable the line refers to, by using three fields: the first indicates whether it is a Parameter or a Variable; the second reports the label of the variable; and the third indicates, for Variables only, the lag the line refers to. So, for example, the label

Var: K -1

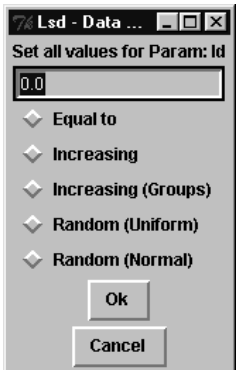
Means that the line contains the values of Variable "K" used at the initial time step as one period lagged value. On the bottom of the window a message will appear when you pass the mouse pointer over one of the label reporting the same label.

The columns refer to the instances of the Object the variables refer to. The column headers report the indexes we have seen above. There are two digits, because each variable instance needs to refer to one Market (first digit) and to one Firm descending from that Market (second digit).

The easiest way to modify the initial values is to simply type the desired values into the cells. A set of bindings allows to quickly step from one cell to the next, by pressing the Enter key, facilitating the data entry. Since the number of instances can also be very high, it is likely that the line's headers disappear while you scroll the window on the right. On the bottom of the window a message reports the instance of the variable the current cell refers to.

For large number of Objects it becomes very tedious to manually insert all the values[11]. It is possible to use of a function that allows to set all the initial values for one variable in all the instances of Objects in the model. On the right of the lines' headers there are buttons labeled "Set All". Pressing on one of these buttons (for example, the one corresponding to "Id") you access a window as shown in Figure 8

Figure 8 – Set All Values



[11] Because of memory constraints of the graphical windows, it is possible to show a maximum of 100 columns, despite the actual total number of instances in the model. For models where some Objects have a higher number of instances it is necessary to use the method described below in order to set the initial values.

This window accepts one numerical value and provides five options. The value inserted is used to set all the values for every variable of that type in the model. The available options are the following, and some of these need also a second value. In this case it is used the number reported in the very first cell in the previous window, that is, the value on the first cell on the right of the button “Set All” you pressed. In the following we refer to “inserted value” to indicate the one typed in the window in Figure 8 and “first value” to indicate the value in the first cell for that variable in the main “Data Editor” window show in Figure 7 above.

The options available are the following:

- Equal To: the inserted value is assigned to all the variables in the model;
- Increasing: the first value is not changed; the values for all other instances are assigned an increasing number, equal to the one placed in the previous cell plus the inserted value.
- Increasing (Groups): the same as before, but instead of assigning continuously increasing values for all the instances in the model, instances are firstly grouped according to the instances of the Objects they descend from. For each first instance in the groups the starting points are re-initialized.
- Random (Uniform): a random value is assigned to any instance. The random numbers are drawn from a uniform distribution whose minimum value is the first value and the maximum is the inserted value.
- Random (Normal): same as above, but the random function is a normal distribution whose mean is the first value and the standard deviation is the inserted value.

Simulation Settings

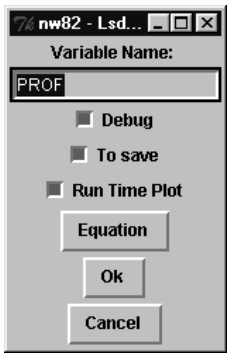
This paragraph describes how to set the options that affect the simulation runs, but are not related to the model, that is:

- variables to save;
- variables to plot run-time;
- number of simulation runs;
- number of steps per simulation run.
- seed for pseudo-random number generator.

Settings on Variables

To save one variable (either a Variable or a Parameter, if that makes sense) you need to point the Browser on the Object containing that Variable. The settings for saving or plotting variables here described are always determined for all the variables of the chosen type in the model. Thus, for example, if the model has 8 Objects Firm, it is not possible to save Variable PROF only for some of the Firms, and not for the others: either all the 8 PROF's are saved or none.

Figure 9 – Variable Options



Double-clicking on the name of the Variable you obtain a window as shown in Figure 9:

- The variable name. This entry allows to edit the name of the Variable. It is used when writing a model to correct possible misspelling of the variables' name. If you change the name of the variable, it will not be found by the equations, and an error will occur when you launch the simulation.
- A checkbox named "Debug": sets the variable's equation to be debugged. Does not affect a normal simulation run (see below for running a simulation in "Debug Mode").
- A checkbox named "To Save": if checked on, save all the variables of this type in the model.
- A checkbox named "Run Time Plot": if checked on, plots the values of these variables during run time.
- The button "Equation" discussed above.

During the simulation run, all the values for the variables checked to be saved are stored the result file. This file is used in the Result Analysis module to plot their time series graphs. This file is a tab delimited text file, whose first line contains the labels of the variable saved (including the index when necessary), and each line refers to a single time step. Hence, it can be easily loaded by a statistical package (like SPSS) to make an accurate study of the simulation result. The same format is used for summary files created when batteries of simulations are used, but the lines will refer to the final step of each simulation run.

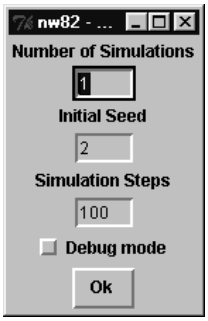
When a simulation is launched, Lsd controls if there are variables to be plotted. If it finds at least one, it creates the run time plot described below. There are two considerations to be made when deciding the variables to plot at run time. First, the mode variables series have to be plotted, the slower is the simulation, thus it is better to reduce at the minimum the number of these variables. Second, the run time plots have the y scale automatically set on the minimum and the maximum values encountered during the simulation run. Hence, if you plot, say, market shares (counted in decimals) and profits (counted in thousands) the y scale will not allow to follow the small variability of the values reported.

If you want to quickly remove all the indications to save or plot variables, there are two entries in menu "Run": "Remove Save Flags" and "Remove Plot Flags"[12]. These functions avoid users to browse through the entire model searching for unwanted settings. Remember that, after having used one of these two entries, no variable will be plotted or saved, unless you re-set some as indicated above.

Other Simulation Settings

Opening the menu "Run" and then choosing the entry "Sim. Settings" a window like the one shown in Figure 10 will appear.

Figure 10 – Simulation Settings



You can set the following preferences:

- Number of simulations. Lsd will run as many simulations as indicated. After each simulation run is terminated, the model is reloaded from the disk as it was in the beginning of the very first simulation run. If this number is higher than 1, Lsd creates also a summary file containing the

[12] There is also the possibility to remove all the debug flags, that disabled every Variable from being debugged.

values of the last time step for each simulation run, besides the individual result files for each simulation run.

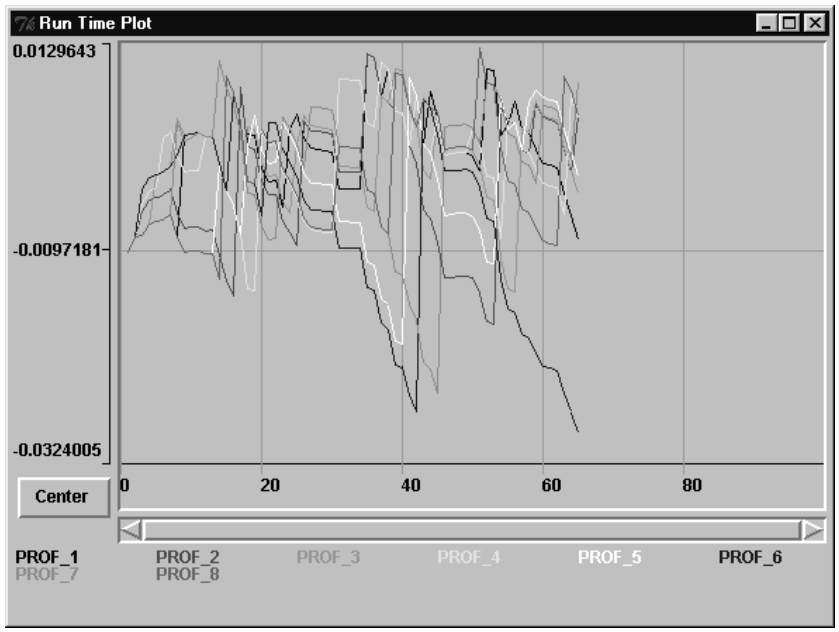
- Initial Seed: Sets a value used a random number generator seeds. For stochastic models, this allows to obtain exactly the same sequence of pseudo-random numbers, if you repeat the simulation with the same seed. If you indicated more than one simulation runs, the seed is changed for each simulation, increasing its value of one. The seed is also used to set the names of the result files (see below).
- Simulation Steps: the number of simulation steps for each simulation run.
- Debug: this checkbox allows to run the simulation in debug mode. See below.

Running a Simulation

All the settings for the simulation (both initial values in the model and the other simulation settings) are stored in the model file. When running a simulation, the model settings currently in memory are written on file, possibly overwriting an existing data file with the same name. Hence, if you made the exercises proposed above, you'd better empty the system (entry "Empty" in menu "File") and re-load it. To run a simulation after having loaded a model' data file is sufficient to open the menu "Run" and choose the entry "Run", accepting all the default settings in the file. The system will present a window summarizing the settings of the simulation and indicating in which files it is going to write the results. There will be as many result files as many simulation runs you chose to run. The names of these files have the first part copied by the name of data file you loaded the data from. To this name is attached a number, indicating the seed generator used for that simulation run. Hence, the names of the result files provide a complete indication of the data used for the simulation run that provided those results. Remember that a simulation run overwrites the data file with the content of the model in memory. If you made some changes to the initial data and you don't want to overwrite the original data file, you need to save the model with a different name, before running the simulation.

During the simulation run the Browser window disappears. If the model has some variables set to be plotted, a new window will appear containing the graph of the variables you chose to observe at run time. The window appears as in Figure 11.

Figure 11 – Run Time Plot



The lowest section contains the list of the variables to plot (note the index used to differentiate among different instances). The variables are written in different colors, corresponding to the same colors of the series in the graph. The horizontal scale depends by the number of steps of the simulation (100 in the Nelson and Winter example). For simulation longer than 600 time steps, the horizontal scrollbar allows to show a portion of the entire plot. During the simulation users can observe any part of the entire window, for example to study the initial stages while the simulation continues, by using the scrollbar below the plot. The button “Center” automatically scrolls the window to show the last points currently plotting.

The vertical scale is automatically adjusted while new values are plotted.

If no variable is checked to plotted during the simulation run, the Log window will print the step just terminated and the number of simulation run currently executed. In any case, both when there is a graph being plotted, the Log window allows to control the simulation while it is running.

Figure 12 – Log Window

There are four buttons that change the type of information provided by the system on the ongoing simulation:

- Stop: interrupt the simulation and return to the Browser.

- Fast: disable all windows, till the simulation is terminated. Note that it still enables the run-time plotting (if any), but it will be mapped on the screen only when the simulation is terminated. Therefore, the use of this button is an intermediate choice, in respect of the speed of execution. The simulation is slowed down because Lsd has to store the information for the run-time plot, but it does not refresh the graph at any time step, but only after the end of any simulation.

- Observe: after having pressed the button “Fast”, this button restores the normal mode of observing the simulation run. Note that, since Fast disables also the Log window, when you press Observe the system will return to refresh the Lsd windows only after the simulation is finished. It is used in case you are running many simulations. You can decide to check the early steps for each simulation, and then let it go quickly to the end. Pressing Observe immediately after Fast will cause the reactivation of the windows at the beginning of next simulation.

- Debug: enable the Debug Mode. Set the simulation run in Debug mode. See below.

Technical Notes: Simulation Runs in Debug Mode

This mode for simulation running is very slow, but allows to follow step by step every single equation computed in the simulation. Model writers can use this option to debug the equation code. When running in debug model, the simulation is stopped any time the equation for one of the Variables checked with the “Debug” option on is just computed. The debugging window shows the

values for all the variables of the model, and users can modify the values, changing the debug option for some instances of Variables individually (with the system saw above, the debug option is enabled or disabled for all the instances of the Varibles). Users can also set conditional debug options for some Variable in the system, which will stop the simulation only if the Variable assumes specific values.

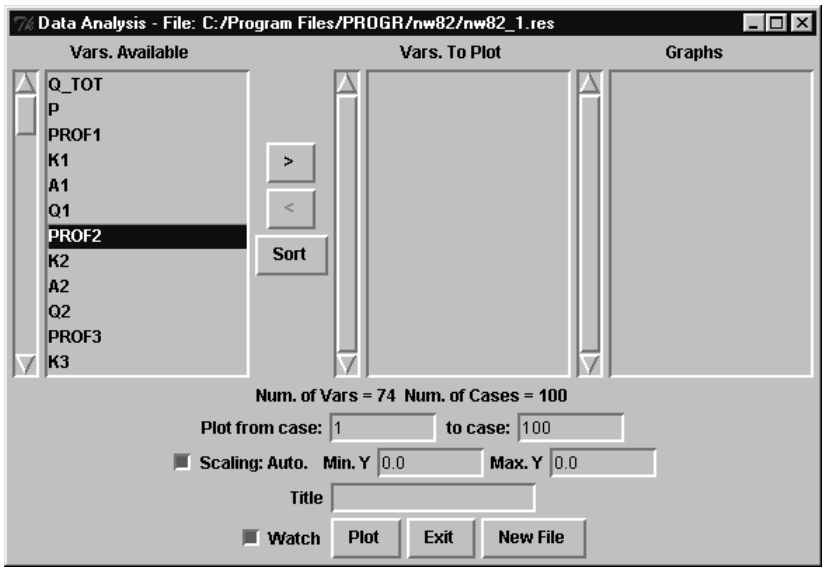
In Debug Mode, the Log window will also print a detailed report of the activities done during the simulation, in order to control the actual sequence of execution for the equations.

Data analysis and Result Files

After a simulation run, you can use the Data Analysis module to plot graphs for all the variables saved during the simulation. The files have the standard extension “res” and are called according to the name of the model and the seed generator used. For example, the model “nw82” (that is, loaded from the file “nw82.res”) launched with seed generator 1, will produce the result file “nw82_1.res”. The name of the result file is also indicated in the confirmation window immediately before the beginning of the simulation.

To load a result file you need to open the entry “Analysis Result” in the menu “Data”. Choose a file (with the default extension “res”) and a window as in Figure 13 will appear.

Figure 13 – Data Analysis



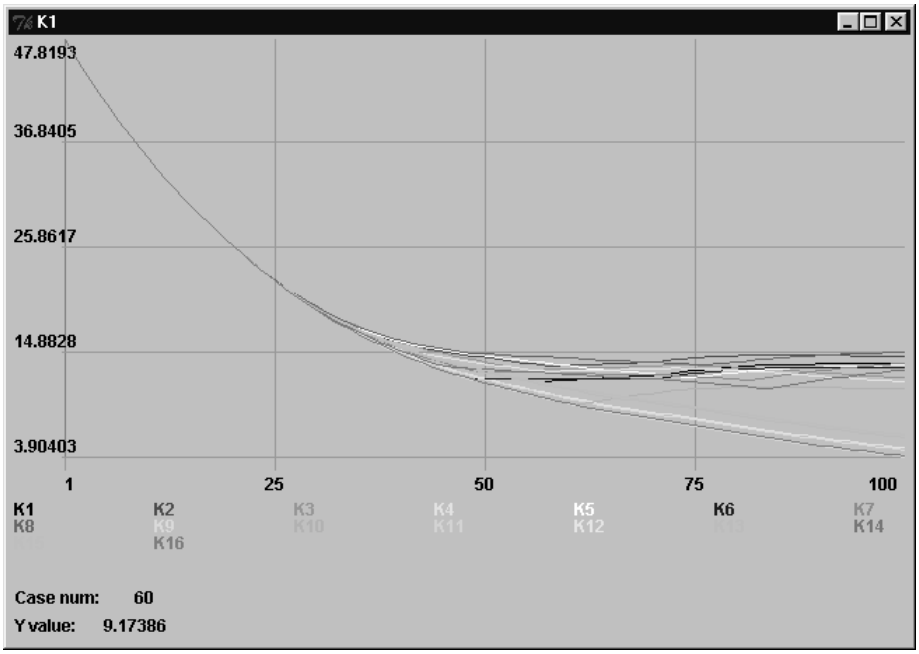
The window contains three boxes: the first on the left contains the whole set of variables saved. You can select some of these variables in many ways. The simplest is to click on one of them and then

clicking on the symbol “>”. You can also double-click and it will immediately be added to the central box. If you select one of the variables, and then click on another one while keeping the key “Shift” pressed, you will also select all the variables in between them. Keeping the key “Control” will instead add to the selection only the ones you clicked on, but will not de-select the previous one. The button labeled “Sort” allows to sort the variables in alphabetical order, so to facilitate the selection of the instances of the same variable.

When you have selected some variable in the middle box, you can directly press “Plot”, and the system will automatically generate a plot.

It will make a series of default choices. The title of the new graph is the name of the first variable you selected. You could have chosen another title by typing it in the entry on the right of the label “Title”.

Figure 14 – Post-Simulation Graph



By default, the graph includes all the cases saved in the file. For very long simulations (more than 600 time steps), the system shrinks the actual cases by making an average of group of points in order to fit the graph with the available dimensions of the window. You can decide to plot only some of the cases to avoid this reduction, by indicating the first and the last case to plot in the relative entries.

By default, the system sets the vertical scale very close to the maximum and the minimum values for all the variables selected. You can decide to determine manually maximum and minimum values

for the vertical axis by un-checking the box on the left of the label “Scaling: Auto” and setting in the two entries the desired min and max values.

It is not possible to directly observe the values of the result files (you can always use a spreadsheet for this, given the standard format of these files). But moving the mouse pointer over a graph makes appear, on the bottom left corner, the number of the case and the y coordinate for that graph corresponding to the point indicated by the mouse.

The produced graphs are listed in the last box on the right. Two properties of these windows facilitate the management of even a large number of graphs. Double-clicking on one of the titles in the third box brings immediately the related graph on the foreground. Instead, clicking on any part of any graph brings the main Data Analysis window in the foreground.

Conclusions

This document presents the usage of the Lsd system to run simulation models. Lsd allows to explore the graphical structure of the model, observe and edit both the number of entities and the initial values for a simulation model only by using user-friendly graphical interfaces. Users can also easily choose run time settings like number of simulation steps, pseudo-random number seed generators, number of simulation runs. Models’ results can be plotted on run-time graphs during the simulation run or analyzed with post-simulation study of any variable in the model.

Hence, a model written in Lsd is easily explored thoroughly, given the opportunity of easily test its behavior with any setting. Moreover, it gives the opportunity to a wide audience of unskilled computer users to make use of simulation models.

All the functions of the systems are provided automatically by Lsd when the model is created, without need for model writers to worry for any technical detail of the simulation model not directly related with the model content. Though not discussed in depth, few hints on the methodology to be used to write models for Lsd are discussed. In particular, it is underlined that model writers have only to write C++ code for their equations. Each equation is written individually, and the references to other parts of the model are made only via variable labels. Thus, the programming knowledge required to write a simulation model is minimal, and only dependent on the computational complexity of the model itself, which, moreover, is reduced by the availability of a Lsd library of functions. Moreover, the resulting models are basically composed by C++ code, ensuring high speed of execution and portability on different platforms. The Lsd system is a freeware code

distributed in two versions for Windows 95 and Unix. It is written in C++ (Borland C++ for Windows 95 and GNU gcc for Unix) and Tcl/Tk.

Danish **R**esearch **U**nit for **I**ndustrial **D**ynamics

The Research Programme

The DRUID-research programme is organised in 3 different research themes:

- The firm as a learning organisation

- Competence building and inter-firm dynamics

- The learning economy and the competitiveness of systems of innovation

In each of the three areas there is one strategic theoretical and one central empirical and policy oriented orientation.

Theme A: The firm as a learning organisation

The theoretical perspective confronts and combines the ressource-based view (Penrose, 1959) with recent approaches where the focus is on learning and the dynamic capabilities of the firm (Dosi, Teece and Winter, 1992). The aim of this theoretical work is to develop an analytical understanding of the firm as a learning organisation.

The empirical and policy issues relate to the nexus technology, productivity, organisational change and human ressources. More insight in the dynamic interplay between these factors at the level of the firm is crucial to understand international differences in performance at the macro level in terms of economic growth and employment.

Theme B: Competence building and inter-firm dynamics

The theoretical perspective relates to the dynamics of the inter-firm division of labour and the formation of network relationships between firms. An attempt will be made to develop evolutionary models with Schumpeterian innovations as the motor driving a Marshallian evolution of the division of labour.
The empirical and policy issues relate the formation of knowledge-intensive regional and sectoral networks of firms to competitiveness and structural change. Data on the structure of production will be combined with indicators of knowledge and learning. IO-matrixes which include flows of knowledge and new technologies will be developed and supplemented by data from case-studies and questionnaires.

Theme C: The learning economy and the competitiveness of systems of innovation.

The third theme aims at a stronger conceptual and theoretical base for new concepts such as 'systems of innovation' and 'the learning economy' and to link these concepts to the ecological dimension. The focus is on the interaction between institutional and technical change in a specified geographical space. An attempt will be made to synthesise theories of economic development emphasising the role of science based-sectors with those emphasising learning-by-producing and the growing knowledge-intensity of all economic activities.

The main empirical and policy issues are related to changes in the local dimensions of innovation and learning. What remains of the relative autonomy of national systems of innovation? Is there a tendency towards convergence or divergence in the specialisation in trade, production, innovation and in the knowledge base itself when we compare regions and nations?

The Ph.D.-programme

There are at present more than 10 Ph.D.-students working in close connection to the DRUID research programme. DRUID organises regularly specific Ph.D-activities such as workshops, seminars and courses, often in a co-operation with other Danish or international institutes. Also important is the role of DRUID as an environment which stimulates the Ph.D.-students to become creative and effective. This involves several elements:

- access to the international network in the form of visiting fellows and visits at the sister institutions
- participation in research projects
- access to supervision of theses
- access to databases

Each year DRUID welcomes a limited number of foreign Ph.D.-students who wants to work on subjects and project close to the core of the DRUID-research programme.

External projects

DRUID-members are involved in projects with external support. One major project which covers several of the elements of the research programme is DISKO; a comparative analysis of the Danish Innovation System; and there are several projects involving international co-operation within EU's 4th Framework Programme. DRUID is open to host other projects as far as they fall within its research profile. Special attention is given to the communication of research results from such projects to a wide set of social actors and policy makers.

DRUID Working Papers

96-1 **Lundvall, Bengt-Åke:** The Social Dimension of the Learning Economy. (ISBN 87-7873-000-7)

96-2 **Foss, Nicolai J.:** Firms, Incomplete Contracts and Organizational Learning. (ISBN 87-7873-001-5)

96-3 **Dalum, Bent and Villumsen, Gert**:Are OECD Export Specialisation Patterns Sticky?' Relations to the Convergence-Divergence Debate. (ISBN 87-7873-002-3)

96-4 **Foss, Nicolai J:** Austrian and Post-Marshallian Economics: The Bridging Work of George Richardson. (ISBN 87-7873-003-1)

96-5 **Andersen, Esben S., Jensen, Anne K., Madsen, Lars and Jørgensen, Martin:** The Nelson and Winter Models Revisited: Prototypes for Computer-Based Reconstruction of Schumpeterian Competition. (ISBN 87-7873-005-8)

96-6 **Maskell, Peter:** Learning in the village economy of Denmark. The role of institutions and policy in sustaining competitiveness. (ISBN 87-7873-006-6)

96-7 **Foss, Nicolai J. & Christensen, Jens Frøslev:** A Process Approach to Corporate Coherence. (ISBN 87-7873-007-4)

96-8 **Foss, Nicolai J.:** Capabilities and the Theory of the Firm. (ISBN 87-7873-008-2)

96-9 **Foss, Kirsten:** A transaction cost perspective on the influence of standards on product development: Examples from the fruit and vegetable market. (ISBN 87-7873-009-0)

96-10 **Richardson, George B.:** Competition, Innovation and Increasing Returns. (ISBN 87-7873-010-4)

96-11 **Maskell, Peter:** Localised low-tech learning in the furniture industry. (ISBN 87-7873-011-2)

96-12 **Laursen, Keld:** The Impact of Technological Opportunity on the Dynamics of Trade Performance. (ISBN 87-7873-012-0)

96-13 **Andersen, Esben S.:** The Evolution of an Industrial Sector with a Varying Degree of Roundaboutness of Production. (ISBN 87-7873-013-9)

96-14 **Dalum, Bent, Laursen, Keld & Villumsen, Gert**: The Long Term Development of OECD Export Specialisation Patterns: De-specialisation and "Stickiness". (ISBN 87-7873-014-7)

96-15 **Foss, Nicolai J.:** Thorstein B. Veblen: Precursor of the Competence-Based Approach to the Firm. (ISBN 87-7873-015-5)

96-16 **Gjerding, Allan Næs:** Organisational innovation in the Danish private business sector. (ISBN 87-7873-016-3)

96-17 **Lund, Reinhard & Gjerding, Allan Næs:** The flexible company Innovation, work organisation and human ressource management. (ISBN 87-7873-017-1)

97-1 **Foss, Nicolai J.:** The Resource-Based Perspective: An Assessment and Diagnosis of Problems. (ISBN 87-7873-019-8)

97-2 **Langlois, Richard N. & Foss, Nicolai J.:** Capabilities and Governance: the Rebirth of Production in the Theory of Economic Organization. (ISBN 87-7873-020-1)

97-3 **Ernst, Dieter:** Partners for the China Circle? The Asian Production Networks of Japanese Electronics Firms. (ISBN 87-7873-022-8)

97-4 **Richardson, George B.:** Economic Analysis, Public Policy and the Software Industry. (ISBN 87-7873-023-6)

97-5 **Borrus, Michael & Zysman, John:** You Don't Have to Be A Giant: How The Changing Terms of Competition in Global Markets are Creating New Possibilities For Danish Companies. (ISBN 87-7873-024-4)

97-6 **Teubal, Morris.:** Restructuring and Embeddeness of Business Enterprises-Towards an Innovation System Perspective on Diffusion Policy. (ISBN 87-7873-025-2)

97-7 **Ernst, Dieter & Guerrieri, Paolo:** International Production Networks and Changing Trade Patterns in East Asia: The case of the Electronics Industry. (ISBN 87-7873-026-0)

97-8 **Lazaric, Nathalie & Marengo, Luigi:** Towards a Characterisation of Assets and Knowledge Created in Technological Agreements: Some evidence from the automobile-robotics sector. (ISBN 87-7873-027-9)

97-9 **Ernst, Dieter.:** High-Tech Competition Puzzles. How Globalization Affects Firm Behavior and Market Structure in the Electronics Industry. (ISBN 87-7873-028-7)

97-10 **Foss, Nicolai J.:** Equilibrium vs Evolution in the Resource-Based Perspective: The Conflicting Legacies of Demsetz and Penrose. (ISBN 87-7873-029-5)

97-11 **Foss, Nicolai J.:** Incomplete Contracts and Economic Organisation: Brian Loasby and the Theory of the firm. (ISBN 87-7873-030-9)

97-12 **Ernst, Dieter & Lundvall, Bengt-Åke:** Information Technology in The Learning Economy – Challenges for Developing Countries. (ISBN 87-7873-031-7)

97-13 **Kristensen, Frank Skov.:** A study of four organisations in different competitive environments. (ISBN 87-7873-032-5)

97-14 **Drejer, Ina, Kristensen, Frank Skov & Laursen, Keld:** Studies of Clusters as a Basis for Industrial and Technology Policy in the Danish Economy. (ISBN 87-7873-033-3)

97-15 **Laursen, Keld & Drejer, Ina.:** Do Inter-sectoral Linkages Matter for International Export Specialisation? (ISBN 87-7873-034-1)

97-16 **Lundvall, Bengt-Åke & Kristensen, Frank Skov.:** Organisational change, innovation and human ressource Development as a response to increased competition. (ISBN 87-7873-036-8)

98-1 **Præst, Mette.:** An Empirical Model of Firm Behaviour: A dynamic Approach to Competence Accumulation and Strategic Behaviour. (ISBN 87-7873-037-6)

98-2 **Ducatel, Ken.:** Learning and skills in the Knowledge Economy. (ISBN 87-7873-038-4)

98-3 **Ernst, Dieter.:** What Permits Small Firms to Compete in High-Tech Industries? Inter-Organizational Knowledge Creation in the Taiwanese Computer Industry. (ISBN 87-7873-039-2)

98-4 **Christensen, Jens Frøslev.:** The Dynamics of the Diversified Corporation and the Role of Central Management of Technology. (ISBN 87-7873-040-6)

98-5 **Valente, Marco.:** Laboratory for Simulation Development. (ISBN 87-7873-041-4)

Information for subscribers.
Subscription price for 1997 is 600 DKR (about 20 papers). The rate for single issues is 40 DKR. It is possible to make a commitment to an exchange of papers from related departments or research teams. All correspondence concerning the DRUID Working Papers should be send to.

Pernille Wittrup
Fibigerstræde 4
DK-9220 Aalborg OE
Tel. 45 96 35 82 65
Fax. 45 98 15 60 13
E-mail: druid-wp@business.auc.dk