

Computer Sciences for Economists

The goal of the course is to teach students how to use computers to make (some) sense of economic phenomena.

In practice, the course will teach how to **design, build, use** and **present** computer simulation models. Though there will be the need to write programming code, the focus is on the ways to obtain information from simulation models.

Introduction to Computer Simulations

This lesson will discuss the basic elements of a simulation model. We will use an extremely simple example both for the presentation and for the first exercise.

The lessons of the course will be partly made of standard presentations and partly made by students' exercises. The students are expected to ask the teacher for help. However, firstly students must try to solve the exercises on their own.

Simulation models

Simulation models are used when we cannot use mathematics, because the idea we have of a model cannot be reduced by usual mathematical simplifications.

Simulation models “simulate” a given reality by producing a series of values for one or more variables. For each time step each variable takes a value produced by an equation.

$$X_t = f(X_{t-1}, Y_t, \alpha, \dots)$$

Simulation models

Let's start with a very simple example. Let's consider the equation:

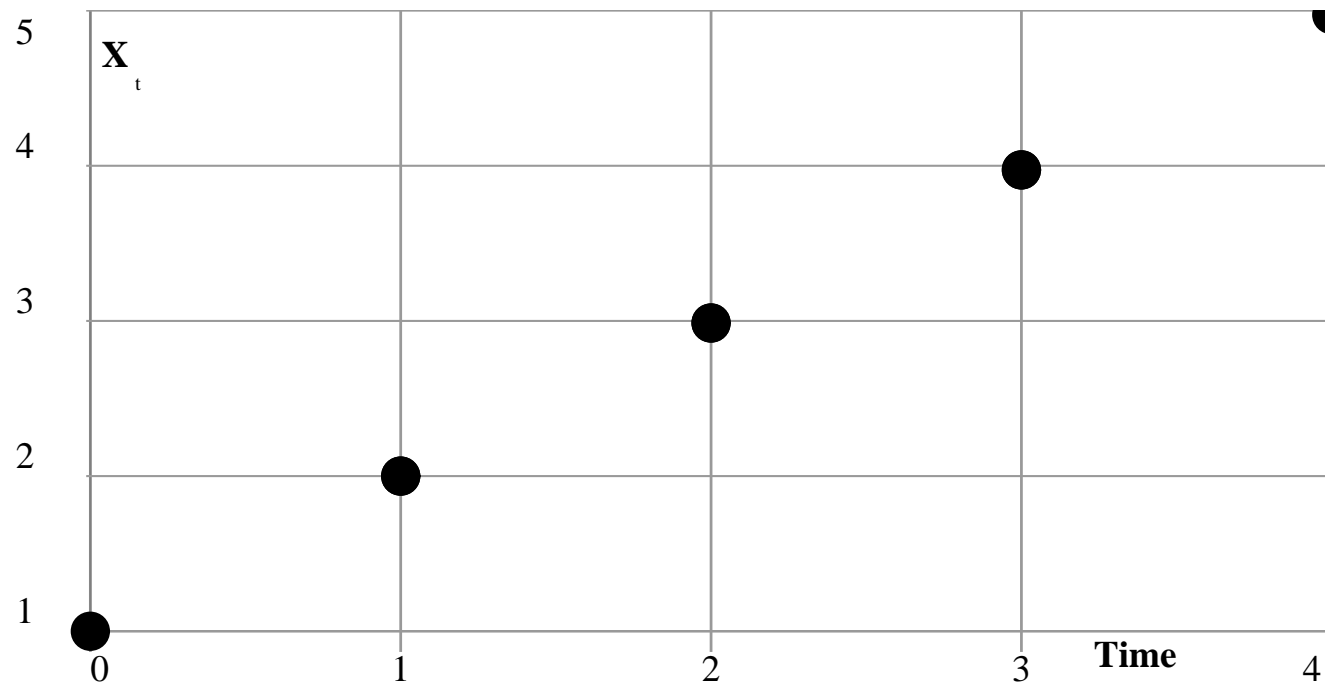
$$X_t = X_{t-1} + 1$$

We will obtain the following values:

X_t	Values
X_0	1
X_1	2
X_2	3
X_3	4
...	...

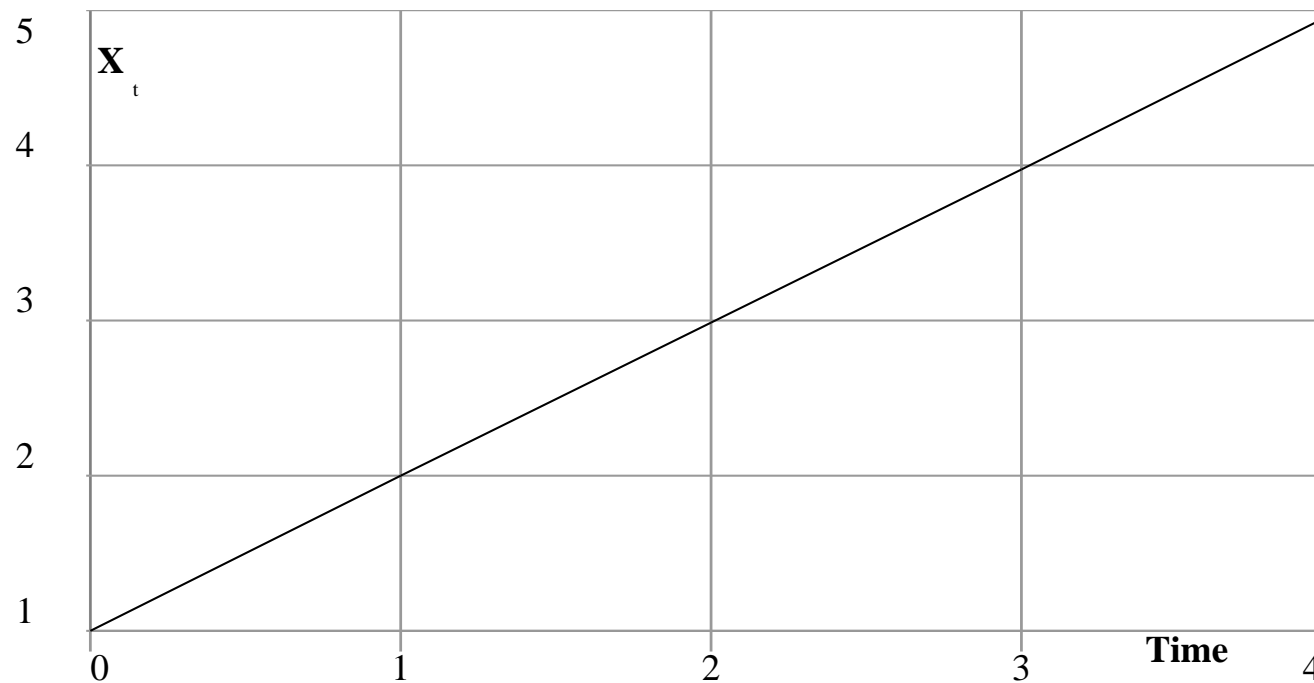
Simulation models

We can express this function using a Cartesian plane, having the time on the X axis and the value of the variable on the Y axis.



Simulation models

We can simplify the reading of the data interpolating lines in between two time points.



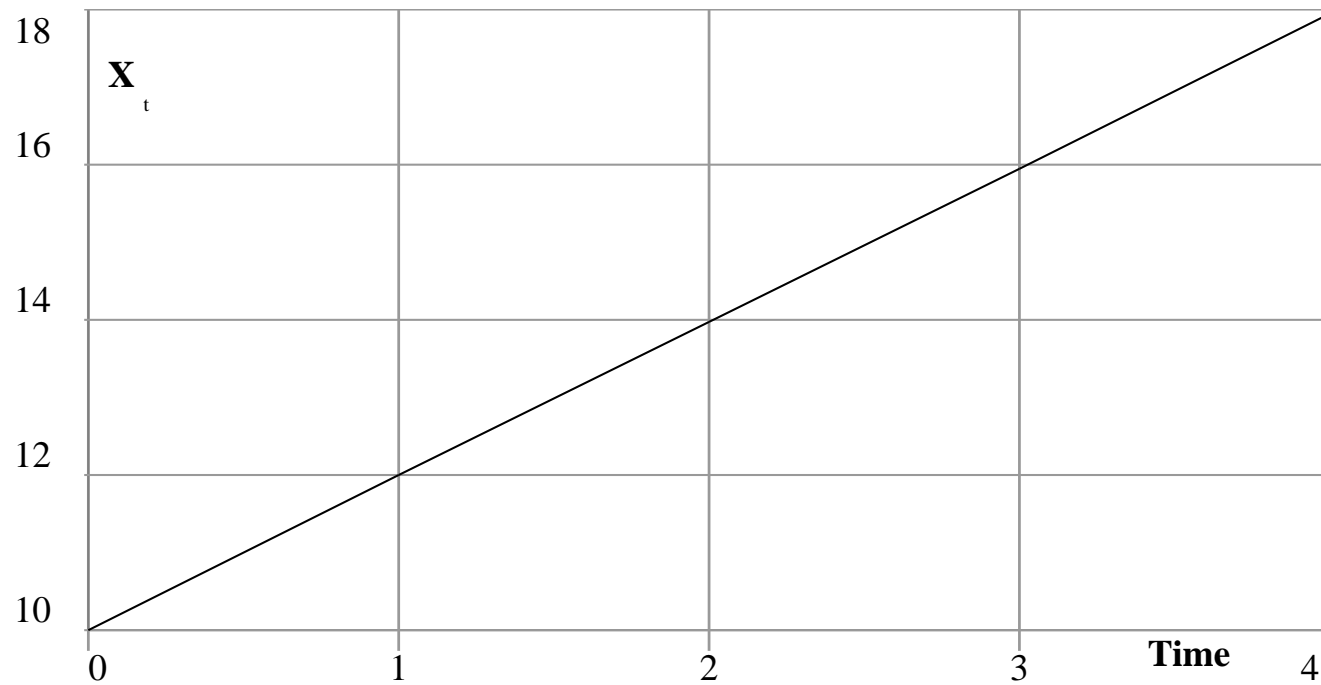
Initial Values

The data produced by a simulation run depend on the equations of the model (in our case $X_t = X_{t-1} + 1$) and by the initial values with which the simulation begins. In our example, at the very first time step we decided to use $X_0 = 1$. Moreover, instead of the 1 we may use any real value. We may refer to this value as the parameter m . Instead of using $m = 1$ and $X_0 = 1$ we can use any value.

In the next example we set $X_0 = 10$ and $m = 2$.

Simulation models

Simulation using $X_0 = 10$ and $m = 2$



Number of Objects

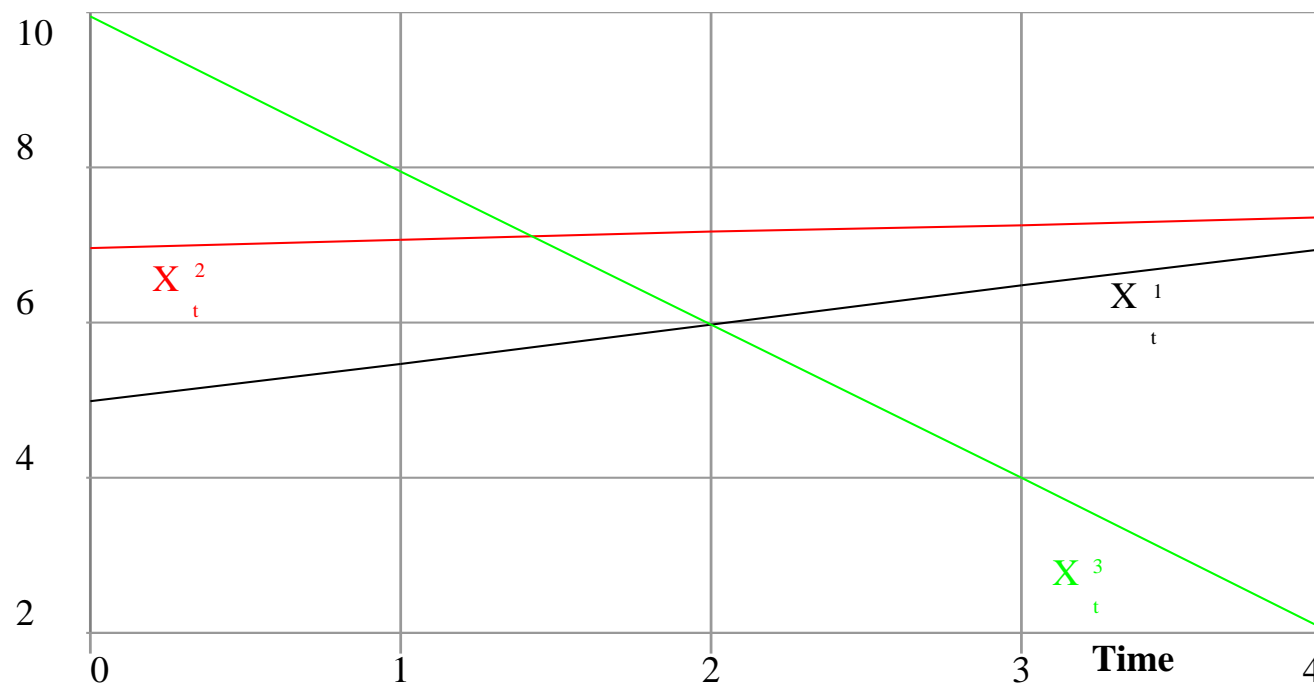
Generally a simulation is used to compare several series computed at the same time. In mathematics it is used the concept of **vectors**. We should have a vector $\vec{X}_t = \{X_t^1, X_t^2, \dots, X_t^m\}$ where each element is computed with the same function, but different parameters $X_t^i = m^i$.

In programming languages it is generally used the concept of **objects**. An object is name (or label) indicating a whole set of variable, parameters, or even other objects. In our example we may define the object $Obj = \{X_t, m\}$. We can then define in our model several objects Obj , and assign to each of the variables and parameters in the different copies different initial values.

In the next example we use three Obj with $X_0 = 5, 7, 10$ and $m = 0.5, 0.1, -2$.

Simulation models

Simulation with three objects *Obj* with initial values: $X_0 = 5, 7, 10$ and $m = 0.5, 0.1, -2$.



Summary: elements of a simulation model

A simulation model is composed by:

- **Variables**, composed by:
 - *Label*: just a name (in the example X);
 - *Equation*: the formula to use to compute its values through time (in the example $X_{t-1} + m$)
 - *Init. values*: (only if used at time $t = 0$) the value to kick-off the simulation (in the example $X_0 = 0$).

Summary: elements of a simulation model

A simulation model is composed by:

- **Variables**, composed by:
 - *Label*: just a name (in the example X);
 - *Equation*: the formula to use to compute its values through time (in the example $X_{t-1} + m$)
 - *Init. values*: (only if used at time $t = 0$) the value to kick-off the simulation (in the example $X_0 = 0$).
- **Parameters**, composed by:
 - *Label*: just a name (in the example m);
 - *Init. values*: the value of the parameter (in the example $m = 1$).

Summary: elements of a simulation model

A simulation model is composed by:

- **Variables**, composed by:
 - *Label*: just a name (in the example X);
 - *Equation*: the formula to use to compute its values through time (in the example $X_{t-1} + m$)
 - *Init. values*: (only if used at time $t = 0$) the value to kick-off the simulation (in the example $X_0 = 0$).
- **Parameters**, composed by:
 - *Label*: just a name (in the example m);
 - *Init. values*: the value of the parameter (in the example $m = 1$).
- **Objects**: the containers of any other element of the model (in the example Obj).

Summary: elements of a simulation model

- **Variables**, composed by:
 - *Label*: just a name (in the example X);
 - *Equation*: the formula to use to compute its values through time (in the example $X_{t-1} + m$)
 - *Init. values*: (only if used at time $t = 0$) the value to kick-off the simulation (in the example $X_0 = 0$).
- **Parameters**, composed by:
 - *Label*: just a name (in the example m);
 - *Init. values*: the value of the parameter (in the example $m = 1$).
- **Objects**: the containers of any other element of the model (in the example Obj).
- **Sim. settings**: num. of steps, values to save (4 and save the X 's).

Languages to implement a simulation model

Simulation models can be implemented not only with computers but with any type of language, **if the elements of the model are defined in detail**. For example, one can use paper and pencil, a pocket calculator, a spreadsheet, or simply reasoning.

A language allows the model to interact with the user, and to users to explain the model to each other.

Any language can be used, as long as one is able to define unambiguously the model's element. Computers are particularly suited because:

- Computers need to be instructed very precisely.
- They can manage reliably large amounts of data very quickly.

However, a model producing tons of data in a nanosec is useless, unless a user can make sense of the model's behaviour.

Computer Languages for Simulation Models

A language serves not only to represent the model (i.e. define variables, parameters, equations etc.) and to execute a run. There are other functions that are necessary for a model to be of any use:

- Grammar for the equations (symbols for the algebraic function);
- Interfaces to implement the model's elements and initializations;
- Interfaces to read the data produced;
- Interfaces to interpret the model's behaviour;
- Interfaces to modify initial data;
- Interfaces to *debug* the model;
- Documentation of the model.

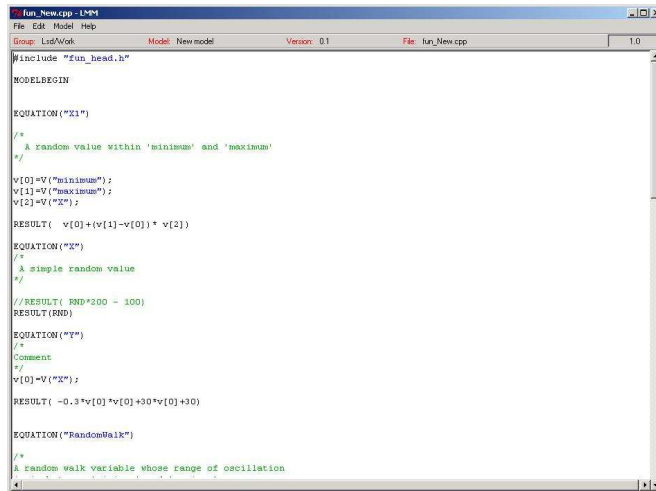
Laboratory for Simulation Development - Lsd

We are going to use a simulation language developed on purpose for building, using and transferring simulation models.

The **equations** of a model in Lsd are written as C++ code, generally using **Lsd Model Manager - LMM**. The C++ code is simply a text containing commands understood by a program which creates new programs: the **compiler**.

The code for the equations is compiled and inserted in a **Lsd model program**, together with all the interfaces needed to use a simulation model. A Lsd model program is used to create the other model elements, setting the initial values, etc. All these elements are saved in a **configuration file**, storing the information about the model. A Lsd model program generally creates and uses many configuration files for different initializations.

Lsd Model Manager - LMM



```
fun_New.cpp - LMM
File Edit Model Help
Group: Lsd/Work Model: New model Version: 0.1 File: fun_New.cpp 1.0

#include "fun_head.h"
MODELBEGIN

EQUATION("X1")
/*
A random value within 'minimum' and 'maximum'
*/
v[0]=V("minimum");
v[1]=V("maximum");
v[2]=V("X");
RESULT( v[0]+(v[1]-v[0])* v[2])

EQUATION("X")
/*
A simple random value
*/
//RESULT( RND*200 - 100)
RESULT(RND)

EQUATION("Y")
/*
Comment
*/
v[0]=V("X");
RESULT( -0.3*v[0]*v[0]+30*v[0]+30)

EQUATION("RandomWalk")
/*
A random walk variable whose range of oscillation
*/
```

Model Equation file

"fun_mymodel.cpp"

*Grammar
Error Messages*

Failure

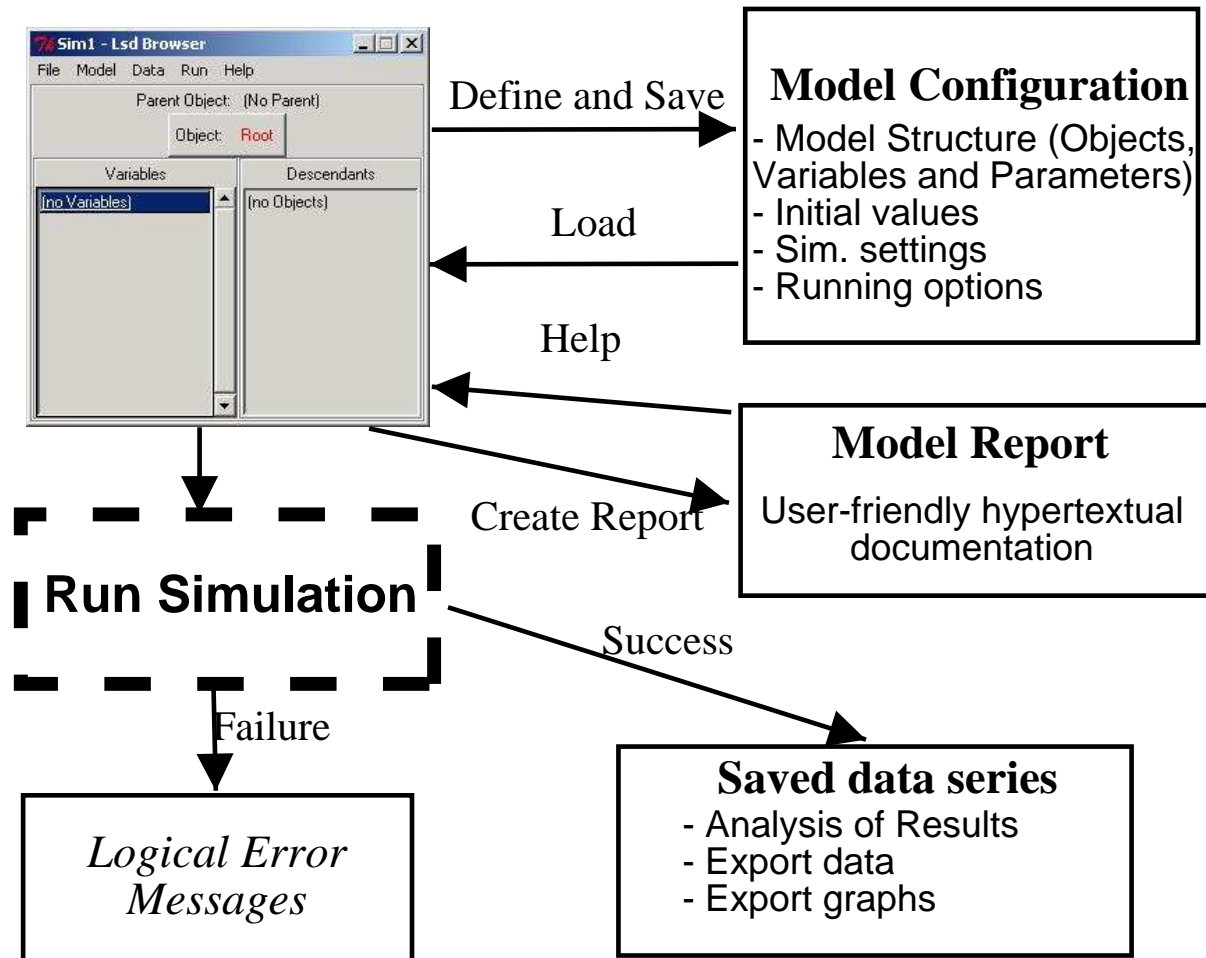
Lsd system code

src\lsdmain.cpp
src\object.cpp
...
src\variable.cpp

Compilation

Success

Lsd Model Program



Simulation 0

Let's implement in Lsd the model $X_t = X_{t-1} + m$. We will implement the following steps:

1. Use LMM to create a new model, call it "Simulation 0".
2. Write the C++/Lsd code for the equation of X .
3. Use the Lsd model program with the equation for X , create:
 - (a) an object Obj ;
 - (b) a variable X in Obj ;
 - (c) a parameter m in Obj ;
4. Set the initial values for the number of copies of Obj , the X_0 's, and m 's.
5. Run a simulation exercise and read the results.

LMM - create a new model

LMM can edit any any text file. However, a user can tell LMM to work on Lsd models. LMM will show the available models, and allow the creation of new ones. To create a new model follow these steps:

1. Browse through the available models.
2. Choose in which group of models to place the new model. Go in the group **Work**.
3. Open menu **Edit/New Model**, and choose **New Model**.
4. Assign a model name “Simulation 0”, a version number, 0.1, and a directory name, S0.
5. Now LMM shows the equation file prepared for the new model.

LMM - The equation file

Any equation file must contain the following text:

```
#include "fun_head.h"
//this is a line comment
/*
This is a multi-line comment
*/
MODELBEGIN //after this line you can insert any equation

MODELEND //never write anything after this line

//ignore the following lines
void close_sim(void)
{

}
```

LMM - The equation file

The equation we need to code is $X_t = X_{t-1} + m$. The minimal expression for this equation in the Lsd equation file is:

```
#include "fun_head.h"
MODELBEGIN

EQUATION("X")
RESULT(VL("X",1)+V("m") )

MODELEND
```

Let's see the meaning of the commands used in the Lsd expression of the equation.

LMM - The equation file

In Lsd the equations are written as separate blocks of code beginning with `EQUATION('X')` and ending with `RESULT(anyValue)`. These lines correspond to the expression:

$$X_t = anyValue$$

Obviously, within the header and closing commands for an equation the modeller can insert any legal C++ code, or a Lsd command.

LMM - The equation file

The most frequently used Lsd command is: $V('Y')$, where Y can be the label of a variable, parameter or function. This command searches in the model the item with that label specified and returns its current value. It is, therefore, the equivalent of Y_t placed on the right side of the equation.

The command has also variations. The one we need is $VL('Y', lag)$ that returns the value of the item lag periods ago, that is Y_{t-lag} . Note that $V('Y') = VL('Y', 0)$.

There are many Lsd commands available for common computational structure. See the LMM help pages on the Macros for Lsd Equations.

LMM - The equation file

The equation's code we wrote earlier was called “minimal” because, though working, is too compact.

Typically, using long variables' labels and with expressions entailing more than a few operators, the expression of the equation within the `RESULT()` becomes impractical.

It is far clearer and more efficient to write the equation over several lines, breaking the computation in small pieces, and storing intermediate results in temporary variables.

The modeller can use the C++ vector `v[0]`, `v[1]`, etc. to store intermediate results. The equation thus become as follow.

LMM - The equation file

The equation $X_t = X_{t-1} + m$ is expressed as:

```
#include "fun_head.h"
MODELBEGIN

EQUATION("X")
/*
The new X is equal to its own past value plus 'm'
*/
v[0]=VL("X",1); //past value of X, lagged of 1 period
v[1]=V("m"); //current value of m
v[2]=v[0]+v[1];
RESULT(v[2] )

MODELEND
```

LMM - The equation file

The C++ local variables $v[0]$, $v[1]$, $v[2]$, etc. are temporary storing places for intermediate calculations. The values stored there can be used only within one execution of the equation, and cannot be used to transfer values across different equations. These variables are created when an equation begins its computation, and are destroyed immediately after its conclusion

Notice also the comments added to the equation. All the text placed by the modeller in the comment after the header of the equation is considered by Lsd the “official” documentation of the variable, and will be automatically included in the documentation of the model.

LMM - The equation file

To write the code for the equations in LMM you can just type the symbols. However, LMM is an editor specifically designed to write code for Lsd models. Many **scripts** make LMM insert the most frequently used code with the user just typing in model specific values. For example, you can insert the lines for
`EQUATION('VarLabel') ... RESULT()` pressing **Control-i** to show the available scripts and **e** for inserting an equation.

LMM - Compiling

Now we need to compile the equation file together with the result of the Lsd source code. This process entails many complicated commands. LMM takes care of this by simply using the menu **Model/Run**.

Writing code (and therefore also simulation models) means to spend 90% of time to fix errors. Here is the first possibility of errors: a wrong command, for example forgetting the semicolon at the end of the lines when requested, prevents the compiler to understand the code, and the compilation process fails. In this case, LMM generates a new window with approximate indications on the type and location of the error.

LMM - Compiling

IMPORTANT: Compiling means creating a new file, in our case `lsd_gnu.exe`. In Windows it is not possible to write on disk a file of a program already running. If you have a Lsd model program already running, the compilation fails even if there are no errors in the code.

In these cases (*very* frequent) you need to close the Lsd model program and recompile the model from LMM with **Model/Run**.

Lsd Browser

When the compilation succeeds LMM runs automatically the Lsd model program. The user interact with the model using the **Lsd Browser**. The Browser shows the content of an object, initially the empty object **Root**, with two lists: one for the variables and parameters of in the object (**Variables**) and the other for the descending objects (**Descendants**).

The menus are:

- **File:** Load and save configuration files.
- **Model:** Create a model structure (define variables, parameters, objects); generate documentation.
- **Data:** Define and observe initial values; analyse or save results.
- **Run:** Define simulation settings and run simulations.
- **Help:** Open help pages on Lsd interfaces and on the specific model.

Generate Model Structure

Let's use the Lsd Browser to generate the model structure. We need to define one object containing a variable and a parameter.

IMPORTANT: to label model entities use only characters (no spaces or other symbols). Elements' labels are case-sensitive.

1. **Model/Add Descending Obj:** Create an object `Obj`.
2. Click on `Obj` in the list of **Descendants**. The Browser shows the new object.
3. **Model/Add a Variable:** Create an variable `X` with 1 lag.
4. **Model/Add a Parameter:** Create a parameter `m`.
5. **File/Save as:** Save the current configuration in a new file `Sim1`.

Generate Initial Data

We now need to determine the initial values. They consists in the number of copies for the only object type in the model, and to insert the values of the X 's at time 0 and for the parameters' m .

1. (with the Browser showing `Obj`) **Data/Set num. objects/All objects** (shortcut `control-o`): there is initially only one copy of `Obj`. Click on the number and insert 10. Press Ok and Exit to return to the **Browser**.
2. (with the Browser showing `Obj`) **Data/Init. values** (shortcut `control-i`): the window shows one cell for each X_0 and for each m , indicated by an indicator for the copy of `Obj`. Use **Set All** to generate all X equal to 10 and the m ranging from -5 to 4.

Help yourself

All the windows have pretty intuitive graphical interfaces. Moreover, each window has a button (or menu item) for help. Use these help pages for understanding how to operate the windows. Notice that the help pages are linked to other, related help pages. The whole Lsd hyper-manual is organized for the main menu items.

In general, the windows allow quite elaborated operations, but the default choices are the most frequently used ones.

Simulation Settings

We are now able to run the simulation, but that would be useless. In fact, Lsd automatically discards the values of the variables as soon as they are not necessary for the computations of the model. Users must specifically instruct the model to save the series of the variables relevant to be analysed.

Move the Browser on `Obj` and double-click on `X`. The system will show the options available for the variable. Select the options **Save** and ensure it is checked on.

We have not told the system to store the values for all the copies of `X`'s to be saved.

Simulation Settings

There are other settings relevant for a simulation run that do not concern directly the model content. Choose menu **Run/Sim.Setting**. The window will allow to set several options concerning how to manage simulation runs. Ignore all of them and assign 10 to the number of steps, replacing the default value of 100.

Simulation run

We can now run the simulation. Choose menu **Run/Run** and a summary window will appear.

This window reminds the user that the configuration defining the model, initial values, options, etc. is going to be saved in a file before starting the actual simulation. Any file with the same name possibly present will be cancelled and overwritten.

Saving the configuration allows the user to be able to perfectly replicate the last simulation, so that errors or unexpected results can be replicated and investigated. To avoid overwriting a file it is sufficient to change name of the configuration.

Confirm and the simulation will start, finishing shortly after.

Simulation run

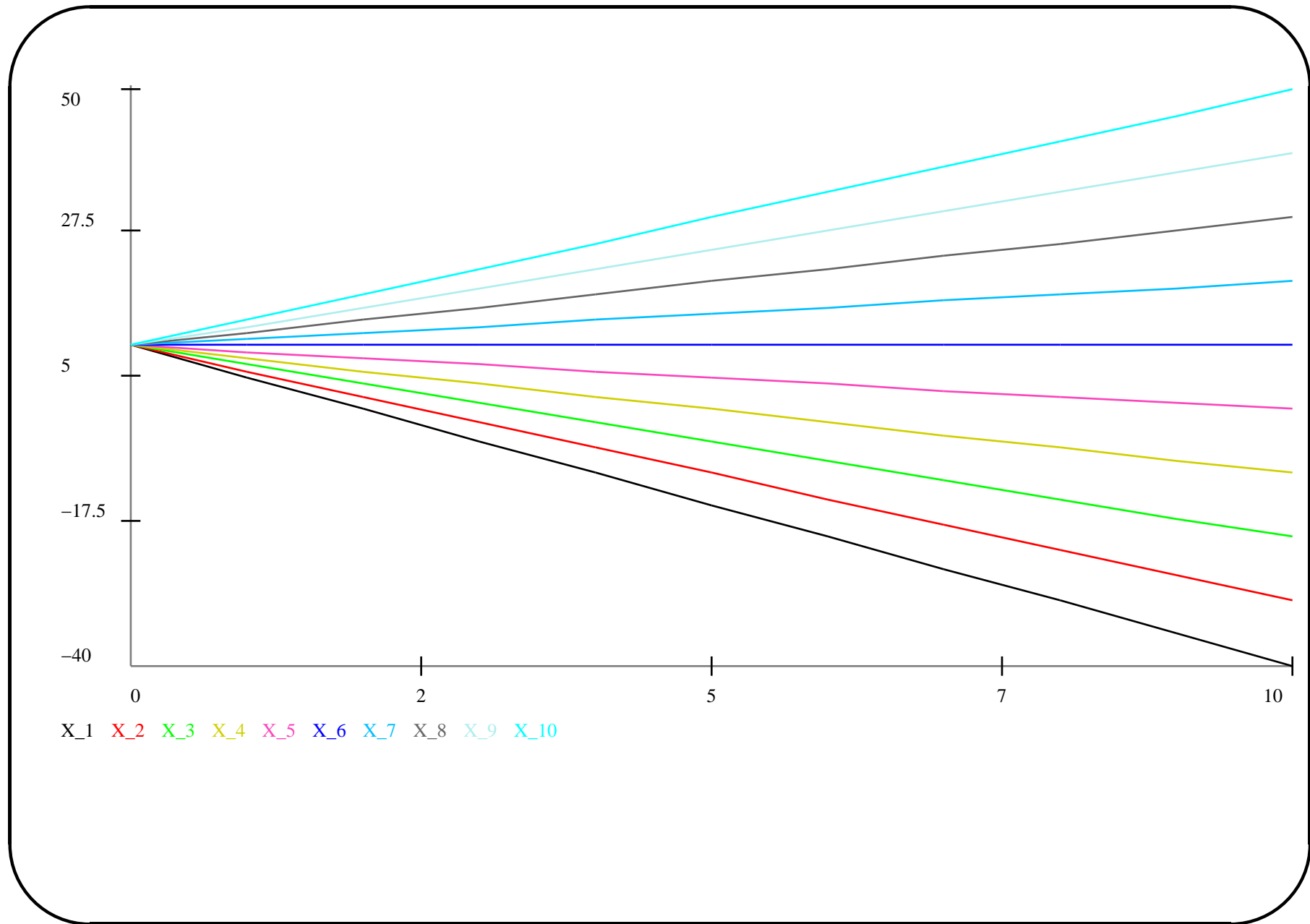
After a simulation run the browser will re-appear. Though nothing seems changed as compared to before running the simulation, the Lsd model program differs in its internal content: the initial configuration has been replaced by the configuration of the model at the end of the simulation run.

The system forbids to use this configuration to start directly a new simulation (it would overwrite a file meant to contain the initial configuration). Try to choose again **Run/Run**, and an error message will appear. Press **Cancel** to return.

Analysis of Results

After a simulation we can observe all values of the model that have been saved. In our example all the series X .

1. Open menu **Data/Analysis Results**. The Browser disappears showing a module to analyse
2. Select all series from the list **Series Available**. Press **Plot**. You need to obtain the graph as in the following slide.



Summary

The operations we have performed in this lesson are:

1. Design a model on paper;
2. Writt the code implementing the equation (and fixed bugs in the code?);
3. Define the model structure and initialization;
4. Run the simulation;
5. Analyse the results.

Homework

- Test the *Simulation 0* model with different initializations, number of Obj and number of steps. Check that it does what it is supposed to do.
- Create a new model with the following equation:
$$X_t = m * X_{t-1} * (1 - X_{t-1}).$$
 Test the model for initial values of X_0 between 0 and 1, and for values of m between 1 and 4. Pay particular attention to the values of m around 3, 3.45, 3.547, 3.84.