LSD- Laboratory for Simulation Development

Lecture 1 - Linear model

We will implement in LSD the model:

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

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:
   1. an object *Obj*;
   2. a variable X in *Obj*;
   3. 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

The latest version of LSD can be downloaded from `https://github.com/marcov64/lsd`.

To start the system run the `run.bat` file (Windows) or the `lmm` executable (MacOS or Linux, see the readme.txt file.

You will launch LMM, an editor specialized to code LSD models and compile LSD model programs.

At start time LMM offers the opportunity to browse the existing models. Choose this option or, if you already started the system, choose the menu entry **Model/Browse Models**.

LMM - create a new model

The models' browser of 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"

MODELBEGIN

MODELEND

void close_sim(void)
{

}
```

The code for the equation needs to be placed in between the `MODELBEGIN` and `MODELEND` lines.

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:

$$X_t = anyValue$$

Within the heading and closing commands for an equation the modeler can insert any legal C++ code or LSD keyword.

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.

The text for the equations can be inserted manually. However, LMM is endowed with specific functions to insert automatically the text most frequently used for LSD equations, speeding up the coding time and reducing the number of typing errors.

These functions are called *scripts*, and when activated insert text in the position of the cursor.

The activation can be performed using the keyboard combination **`Control-i`**, or using the right-button of the mouse, or, finally, selecting the relative entry in the menu **Edit**.

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

The equation file contains the computational content of the model. To turn this into an actual program we need to compile it, together with the rest of the LSD code. Choose the menu entry **Model/Run** to compile the equation

Compilation errors. 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.

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 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`.

Fixing Structural Errors

It is easy to make mistakes in typing the labels of the elements or placing them in the wrong object. To correct any error double-click on the element and select in the resulting window the button **Properties** under name element's label.

There are several options for fixing different types of error. Each option needs to be used independently and confirmed when exiting the properties.

To remove an element altogether assign an empty string to its name.

Generate Initial Data

Any model can start only when the data for its state at time t=1 are assigned. The initial data concern the number of objects, the parameters, and the initial values for the variables expressed with a lag in the equations. In our case: number of copies for the object `Obj`, values of the X at time 0, and 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 almost ready to run the simulation.

LSD automatically discard the values of the variables as soon as they are not necessary for the subsequent computation of the model.

Therefore, users must specifically instruct the model to save the series of the variables they want to analyse as relevant result.

Move the Browser on `Obj` and double-click on X. The system will show the options available for the variable. Select the options **Save for later analysis** and ensure it is checked on. This operation tells the system to store the values for all the copies of X's in memory in order to perform post-simulation analysis.

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

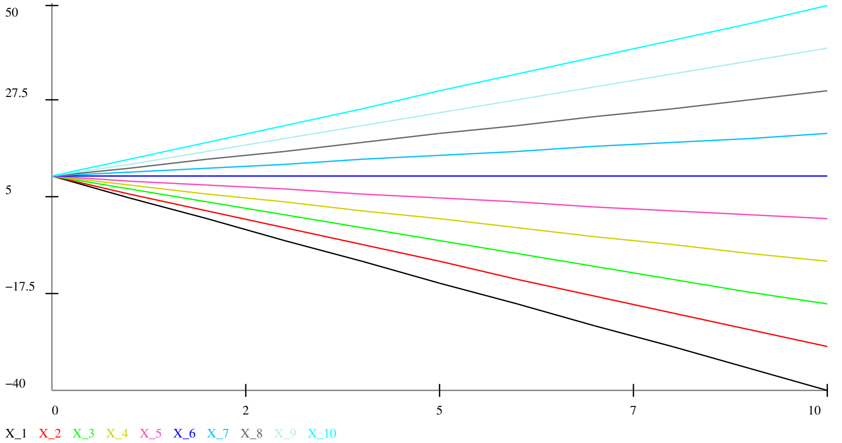
After a simulation run the browser will re-appear. Though nothing seems changed as compared to before running the simulation, the LSD model program differs in its internal content: the initial configuration has been replaced by the configuration of the model at the end of the simulation run.

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 called **Analysis of Results**.
2. Select all series from the list **Series Available**. Press **Plot**. You are expected to obtain a graph as in the following slide.


50
27.5
5
−17.5
−40
0
2
5
7
10
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_10

Summary

The operations we have performed in this lesson are:

1. Design a model on paper;
2. Write the code implementing the equation;
3. Define the model structure and initialization;
4. Run the simulation;
5. Analyse the results.

Exercise

- Test the *Simulation 0* model with different initializations, number of `Obj` and number of steps. Check that it actually provides the expected results.
- Modify the format of the equation, using for example a power function or any other discrete-time dynamics .