ABM Macro Lab: Agent-based Modelling Tools

Session 01

Gabriel Petrini

July, 2025

Outline

Introduction

Objectives

Throughout the sessions, we will

1. Understand how LSD works.
2. Implement a chaotic model[1]
3. Understand and implement the Dosi, Pereira, and Virgillito (2017) model on LSD
4. Analyze the model results on LSD

[1] Based on the slides of LSD manual folder

Structure of the sessions

Session 1 Presents the general structure of LSD and perform some simulations

Session 2 Write the industry model equations in LSD

Session 3 Complete the scripts (if necessary), run the simulation and the analysis of results

Bonus A primer on sensitivity analysis

Repository structure

This presentation can be found on this repository with another examples:

`https://github.com/gpetrini/CodingSession`

Linear_Model/fun_linear.cpp Starting script for the linear model

Chaotic_Model/fun_chaotic.cpp Starting script for the chaotic model

Industry_SummerSchool/fun_industry.cpp Starting script for the industry model

Sim*.lsd Are the configuration files

Important: Clone/Download this folder under `LSD/Work/`

Missed something?
`*_complete.cpp` contains the complete equation file for reference.

Crash course on LSD

C++ for LSD I

LSD adds macros to C++. As a consequence, some basic knowledge on C++ is usefull. What do we need to know here?

In python, R

```
x = 5 # A comment
y = 2.5
## We do not need to
## work with pointers
```

In C++

```
int x = 5; // A comment
double y = 2.5;
object *agent; // Latter
```

Compilation

Different from python and R, we need to compile our code before using it.

C++ for LSD II

To avoid initializing everything, LSD has some already initialized variables:

```
v[0] = 10; // We can assing number to
v[1] = 50; // a vector (always available)
T; // Current simulation time
i = 1; // i,h,j,k can be used for integers
cur; cur1; // Pre-allocated pointers
```

Those are local variables that we can use on equations.

Debbuging
LSD has an internal debbuger that allow us to look to local variables on the fly.

Macros I

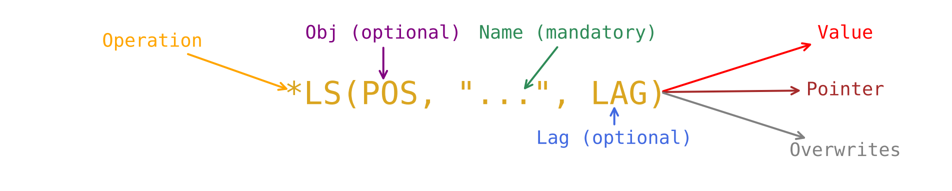
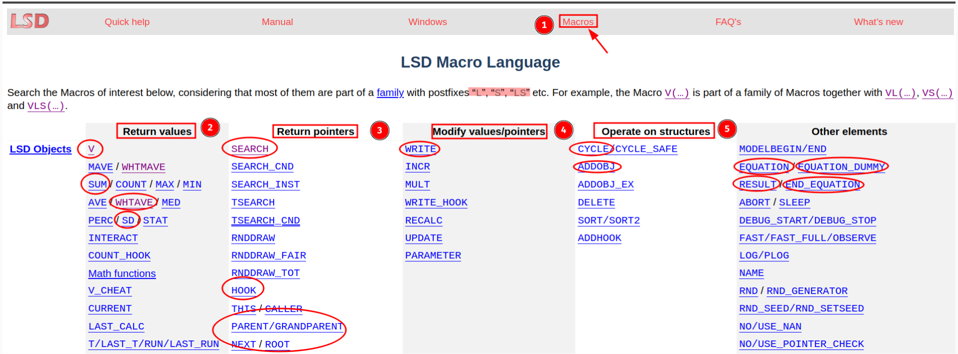


Figure 1: General structure of most LSD macros

- `VLS(POS, "NAME", 1)` Returns the value of `NAME` at lag 1, position `POS`
- `SUMS(POS, "NAME")` Sums the value of variable `NAME` at position `POS`
- `SEARCHS(PARENT, "AGT")` Searches for agent `AGT` starting from `PARENT`
- `WRITELS(POS, "NAME", 1)` Overwrites `NAME` at lag 1 at position `POS`

Macros II



LSD Quick help Manual Windows Macros FAQ's What's new

LSD Macro Language

Search the Macros of interest below, considering that most of them are part of a family with postfixes "L", "S", "LS" etc. For example, the Macro V(...) is part of a family of Macros together with VL(...), VS(...) and VLS(...).

	Return values	Return pointers	Modify values/pointers	Operate on structures	Other elements
LSD Objects	V	SEARCH	WRITE	CYCLE/CYCLE_SAFE	MODELBEGIN/END
	MAVE / WHTMAVE	SEARCH_CND	INCR	ADDOBJ	EQUATION / EQUATION_DUMMY
	SUM / COUNT / MAX / MIN	SEARCH_INST	MULT	ADDOBJ_EX	RESULT / END_EQUATION
	AVE / WHTAVE / MED	TSEARCH	WRITE_HOOK	DELETE	ABORT / SLEEP
	PERC / SD / STAT	TSEARCH_CND	RECALC	SORT/SORT2	DEBUG_START/DEBUG_STOP
	INTERACT	RNDDRAW	UPDATE	ADDHOOK	FAST/FAST_FULL/OBSERVE
	COUNT_HOOK	RNDDRAW_FAIR	PARAMETER		LOG/PLOG
	Math functions	RNDDRAW_TOT			NAME
	V_CHEAT	HOOK			RND / RND_GENERATOR
	CURRENT	THIS / CALLER			RND_SEED/RND_SETSEED
	LAST_CALC	PARENT/GRANDPARENT			NO/USE_NAN
	T/LAST_T/RUN/LAST_RUN	NEXT / ROOT			NO/USE_POINTER_CHECK

Figure 2: Other LSD macros

Macro example I

How can we write the following equation using LSD syntax?

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

```
EQUATION("X") // This is a single-line comment
/*
This is a multi-line comment
*/
v[0] = VL("X",1); //past value of X, lagged of 1 period
v[1] = V("m"); //current value of m (variable or parameter)
v[2] = v[0] + v[1]; // v[n] are local variables
RESULT(v[2]) // Specify the output of the function
```

Variable or parameter?
As a rule of thumb, variables have an `EQUATION` associated, parameters do not.

Macro example II

How can we write the following equation using LSD syntax?

$$Y_t = \sum_{i=1}^{n} X_{n,t} + W_{n,t-1}$$

```
EQUATION("Y")
v[0] = 0; // Initialize the Accumulation
CYCLE(cur, "Firm") { // Similar to a for-loop in other languages
    v[1] = VS(cur, "X"); // cur points to a "Firm" object
    v[2] = VLS(cur, "W", 1); // cur is a locally defined pointer
    v[3] = v[1] + v[2];
    v[0] = v[0] + v[3];
}
RESULT(v[0])
```

Macro example III

We can also write on an alternative way

$$Y_t = \sum_{i=1}^{n} X_{n,t} + W_{n,t-1}$$

```
EQUATION("Y")
// If X and W are bellow Y on the tree (later)
v[0] = SUM("X");
v[1] = SUML("W", 1);
v[2] = v[0] + v[1];
RESULT(v[2])
```

Equation file

Any equation file (.cpp) must contain the following text:

```
// File created using org-mode tangle feature.
#include "fun_head.h" // This is mandatory

MODELBEGIN

// Your code goes here

MODELEND
void close_sim(void) {}
```

In our session, we will copy-and-paste the initialization equation and continue from there.

Model structure and data organization I

Besides the equation files (`.cpp` or `.h`), LSD defines the model structures on a different file (extension `.lsd`). This special file has:

- Variables and parameters names
- Model structure (where elements are)
- Initial and parameter values
- Simulation settings
- Number of objects
- Variables to plot, analyze, debug, etc

LSD and OOP
This structure ensure the modeler to think in terms of data structure.

Logistic chaotic model

Start

Let's start with the file on `Chaotic_Model/fun_chaos.cpp`

```
#include "fun_head.h" // This is mandatory

MODELBEGIN

// Your code goes here

MODELEND
void close_sim(void) {}
```

Equation

Consider the model made of the single equation

$$X_t = m \cdot X_{t-1} \cdot (1 - X_{t-1})$$

To implement this model follow this steps:

1. Insert the equation's code for the model.
2. Compile and run the model (menu Model/Run).
3. Using the Lsd model program generate one object and place in it variable X with 1 lag and parameter m.

Equation on LSD

The model equation can be written as follows:

$$X_t = m \cdot X_{t-1} \cdot (1 - X_{t-1})$$

```
EQUATION("X")
v[0] = VL("X",1);
v[1] = V("m");
v[2] = v[1]*v[0]*(1-v[0]);
RESULT(v[2])
```

Configuring the model structure

- Compile (`Model > Compile and run ...`).
- Wait to open a new window.
- Create a descending object called `Obj1` (`Model > Add Object`)
- Double click on `Obj1`
- Add the Variable X (max. lags $= 1$, $X_0 = 0.5$) and Parameter ($m = 2$)
- Save this configuration as `Sim1.lsd`
- Mark X to be saved (`F5`) and run
- Analyze the results

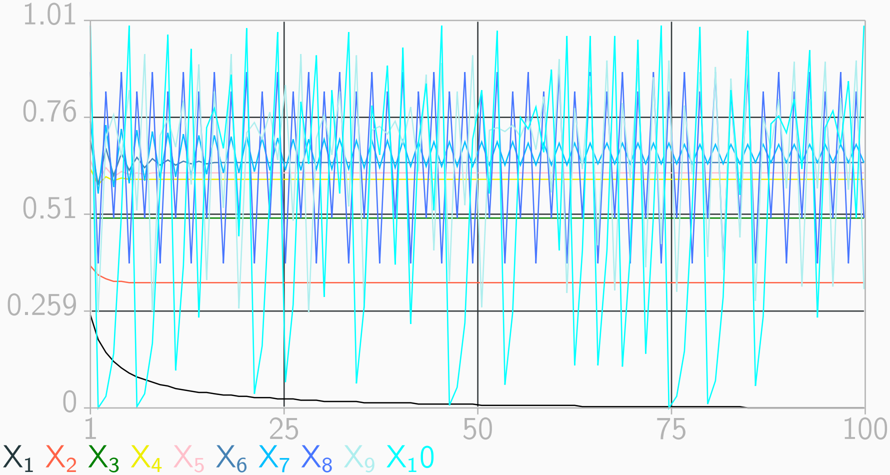
Logistic chaotic model IV

So far, there is nothing new if you use other programming language. However, we can leverage from the fact that LSD is OOP.

- Use menu `Data/Set Number of Objects` and set to 10 the copies of `Obj1`
- Select m and click on `Data>Initial Values ...` and set different values for each instance from $(1 \dots 3.99)$
- Run and plot X

Key takeaways
The source code remains untouched, while we produced a completelly different structure. This is the benefit of isolating the code and the structure.

Results I



Sensitivity to m

The function produces extremely different outcomes depending on the value of m.

- We will create a large number of series computed independently.
- We will set m to a slightly different value for each series.
- We will set the initial points for the X to random values.

Exploiting the chaotic behaviour

- Load `Sim1.lsd` and save it as `Sim2.lsd`
- Generate 10000 copies of `Obj1`
- Open the initial values interface with `Data/Initial Values...`
- Use the buttons `Set All` on the side of m and X to assign values to all the elements.
    - m: Range between 2.8 and 3.99.
    - X: Random (Uniform) between 0.01 and 0.99.
- `Run>Simulation Settings...` to set 1000 time steps.
- Mark m to be saved
- Save, run, analyze

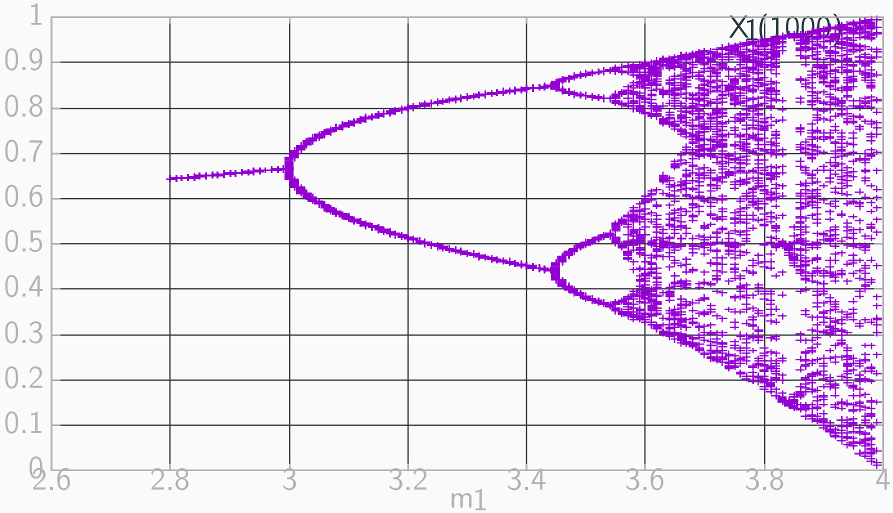
Plotting

- Open Analysis of results
- Select all m and X series
- On the bottom right, select Cross section and XY plot
- Select Points
- Plot for the last timestep (1000, default)
- Hit continue and wait

Key takeaways

Once again, the code remains untouched. Later, we will check other features of LSD that benefit from this design principle.

Results II



The industry model

Where are we and where are we going?

So far, we implemented a small non-economic model. On the following lectures, we will focus on the industry model.

Dosi, Pereira, and Virgillito (2017) design objective: simplest industrial dynamics model capturing most stylized facts:

- Persistent heterogeneity in productivity and all other performance variables
- Persistent market share and entry-exit turbulence
- Skewed size distributions
- Fat-tailed distribution of growth rates
- Scaling of the growth-variance relationship

Equations

Idiosyncratic learning process:	$a_{i,t}$	$=$	$a_{i,t-1} \cdot (1 + \eta \cdot \theta_{i,t})$
Learning shocks	$\theta_{i,t}$	$\sim$	$Beta(\beta_1, \beta_2)$
Market selection	$s_{i,t}$	$=$	$s_{i,t-1} \cdot \left(1 + A \cdot \frac{a_{i,t} - \bar{a}_t}{\bar{a}_t}\right)$
Average productivity	$\bar{a}_t$	$=$	$\sum_{i=1}^{NF} s_{i,t-1} \cdot a_{i,t}$
Exit condition	$s_{i,t}$	$<$	s_{min}
Entrant productivity	$a_{j,t}$	$=$	$\bar{a}_t \cdot (1 + \eta \cdot \theta_{i,t})$
Entrant market-share	$s_{j,t}$	$=$	$1/NF$
Market concentration index	HHI_t	$=$	$\sum_{i=1}^{NF} (s_i)^2$
Market-share adjustment	s_i	$\mapsto$	$s_i \cdot \frac{1}{\sum_{i=1}^{NF} s_i} \Rightarrow \sum_{i=1}^{NF} s_i = 1$
Fixed number of firms	$\#\{1, \ldots, n\}$	$=$	NF

Graph of industry model