Summary

Time now to summarize what are the operations required for a model.

1. Using **LMM/Model Browser** generate a new model. This creates a directory with an equation file.
2. Using **LMM** write the equations for some variables.
3. Using **LMM** compile and run the **Lsd Model Program**. Possibly fix the grammar errors.
4. Using **Lsd** create the objects, variables and parameters. Together with their initialization this form the **model configuration**.
5. Using **Lsd** run the simulations. Possibly fix logical errors.
6. Using **Lsd** analyse the results to make sense of the results.
7. Using **LMM** add new equations for new variables.
8. Using **Lsd** add new variables to the model configurations. Continue from 5.

Summary - Frequent errors

Besides typing errors in the code, the most frequent errors are:

- Using the wrong file for the equations. **LMM** provides always the equation file using the menu **Model/Show Equations**. If you insert the equations in the wrong file, the resulting Lsd model program is not able to compute the value for the variables in the model.
- Mispelling. Names of variables, parameters and objects must be identical, even in their case, in the equations' code and in the model configuration. Such error can cause two types of problems:
  1. An existing variable cannot find its own equation.
  2. The code for an equation cannot find an element in its computation.

  To fix this error you need either to change the label of the entity in the equations or change the label in the configuration.
- Run a simulation just after another run. After a simulation the Lsd model program contains as configuration the last time step. To re-start a new simulation you need firstly to re-load the initial configuration.

B.Arthur's Network Externalities

Standard economics assumes a consumer has personal preferences concerning a product, that are not affected by other events in the economic system. Therefore, at any moment the number and types of product sold in a market indicate the preferences of consumers.

For example, if Microsoft (MS) has 80% of the market and Apple 10% this means that 80% of consumers prefer MS to Apple.

B.Arthur suggested that consumers value some products for two reasons:

1. Their own preferences (as in standard economics);
2. *Network externalities*, that is, the quality added to a product because others use it.

He proposed a model where one can consider explicitly the network externalities effect, showing that the results are quite different.

B.Arthur's Network Externalities

The model considers two groups of consumers, say **type 0** and **type 1**, and two types of product, say **MS** and **Apple**.

A consumer evaluates a product assigning a **utility**, composed by two elements: consumer's own preference and network value of the product.

The **consumers' preferences** are fixed, different for each type or consumer and each product.

The **network value** changes with the number of other users, of whatever type, using that type of product.

B.Arthur's Network Externalities

Let's express formally the utility of consumers. Consider that v_0^{MS} and v_0^{Apple} are the values for consumers 0 if choosing MS or Apple. Moreover, consumers consider relevant in their utility the number of other consumers already using the product. Consider N_{MS} and N_{Apple} these values, and x_0 the network externality effect. For type 1 agent the same symbols use the subscript 1.

$$\text{type } 0 \begin{cases} U_0^{MS} & = v_0^{MS} + x_0 \times N_{MS} & \text{, choosing MS} \\ U_0^{Apple} & = v_0^{Apple} + x_0 \times N_{Apple} & \text{, choosing Apple} \end{cases} \quad (1)$$

B.Arthur's Network Externalities

Equivalently, consumers of type 1 have their utility represented by the following equation:

$$\text{type 1} \begin{cases} U_1^{MS} & = v_1^{MS} + x_1 \times N_{MS} & \text{, choosing MS} \\ U_1^{Apple} & = v_1^{Apple} + x_1 \times N_{Apple} & \text{, choosing Apple} \end{cases} \tag{2}$$

B.Arthur's Network Externalities

When an agent has to choose whether to buy MS or Apple he simply compares the utilities:

$$\text{type 0} \begin{cases} \text{choose MS if} & U_0^{MS} > U_0^{Apple} \\ \text{choose Apple if} & U_0^{MS} < U_0^{Apple} \end{cases} \tag{3}$$

$$\text{type 1} \begin{cases} \text{choose MS if} & U_1^{MS} > U_1^{Apple} \\ \text{choose Apple if} & U_1^{MS} < U_1^{Apple} \end{cases} \tag{4}$$

B.Arthur's Network Externalities

The model works as follows:

1. Create randomly, with 50% probability, whether a new agent is type 0 or type 1.
2. Compute its utilities if choosing MS or Apple.
3. Choose for the new agent the product with the highest probability.
4. Update the N_{MS} if the new agent chose MS and N_{Apple} if chose Apple.

B.Arthur's Network Externalities

The required equations are:

1. Draw randomly a new agent, with probability 50% for each type.
2. given i the type of the new agent, compute U_i^{MS} and U_i^{Apple}
3. choose the product with highest utility.
4. given j is the product chosen, increase the N_j
5. For statistical purposes, compute the share of MS and of Apple as: $S_i = \frac{N_i}{N_{MS}+N_{Apple}}$, where i can be MS or Apple.

Computing the utilities

Let's begin by computing the utilities, for example U_{MS}. This value must be computed using different values depending on whether the new agent is of type 0 or type 1. Therefore, the equation needs to use a conditional statement. In C++, as in any language the conditional statements allow to execute different parts of code depending on a particular condition to be true or false: IF χ THEN (1) ELSE (2). χ must be a condition that can be true or false. If χ is true then the commands (1) are executed and (2) are ignored. Otherwise (1) is ignored and (2) is executed.

Computing the utilities

The equation for the utility of a new agent using MS depends on the network coefficient and on the value of MS for that agent.

```
EQUATION("UtilityMS")
/* Utility in using Microsoft */
v[0]=V("NewAgent"); \\type of new agent
if(v[0]==0)
 {v[1]=V("User0MS"); \\value for 0 in using MS
  v[2]=V("User0Net"); \\net. ext. coefficient for 0
 }
else
 {v[1]=V("User1MS"); \\value for 1 in using MS
  v[2]=V("User1Net"); \\net. ext. coefficient for 1
 }
v[3]=VL("NumMS",1); \\number of existing consumers using MS
v[4]=v[1]+v[2]*v[3];
RESULT(v[4] )
```

Computing the utilities

Equivalently, the utility in using Apple is:

```
EQUATION("UtilityApple")
/* Utility in using Apple */
v[0]=V("NewAgent"); \\type of new agent
if(v[0]==0)
 {v[1]=V("User0Apple"); \\value for 0 in using Apple
  v[2]=V("User0Net"); \\net. ext. coefficient for 0
 }
else
 {v[1]=V("User1Apple"); \\value for 1 in using Apple
  v[2]=V("User1Net"); \\net. ext. coefficient for 1
 }
v[3]=VL("NumApple",1); \\number of existing consumers using Apple
v[4]=v[1]+v[2]*v[3];
RESULT(v[4] )
```

Conditions

Note that in C++ the symbols '=' and '==' mean very different things. '=' is the assignment operator, placing in a variable the value on the right of the symbol. For example, `v[0]=4+5` assigns 9 to `v[0]`.

Instead, '==' is a conditional questions: Is the left-hand-side equal to the right-hand-side? The answer may be TRUE or FALSE. For example, we may write `5+6==3+8` which would always return TRUE. Other conditions are:

- != - Not equal
- >, <, >=, <= - greater , smaller , greater or equal , smaller or equal

Conditions

Moreover, one can link different conditions with the operators AND (expressed as "&&"), OR ("——") and NOT ("!"). For example,

`!(5+6==3+8)` is FALSE;

`(5+6==3+8) AND (2==1)` is FALSE;

`(5+6==3+8) OR (2==1)` is TRUE.

Computing the number of users

The number of consumers using MS will be:

```
EQUATION("NumMS")
/*
Number of MS users
*/
v[0]=VL("NumMS",1);
v[1]=V("UtilityApple");
v[2]=V("UtilityMS");

if(v[2]>v[1])
 v[3]=v[0]+1;
else
 v[3]=v[0];
RESULT(v[3] )
```

Computing the number of users

The number of consumers using Apple will be:

```
EQUATION("NumApple")
/*
Number of Apple users
*/
v[0]=VL("NumApple",1);
v[1]=V("UtilityApple");
v[2]=V("UtilityMS");

if(v[2]>v[1])
 v[3]=v[0]+1;
else
 v[3]=v[0];
RESULT(v[3] )
```

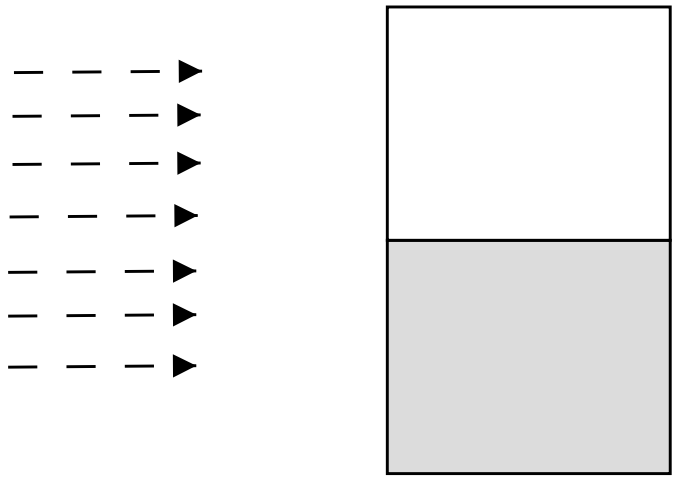
Computing the type of new agent

We are left with the equation for the new agent. We need to have a condition that it provides the results equivalent to flipping a coin. For this we can use the random function `UNIFORM(min, MAX)`. We have seen that this function provides (pseudo-) random values between `min` and `MAX`. Since most frequently models need a random value between 0 and 1, there is a shorter way to write `UNIFORM(0, 1)` We can instead use the command `RND`. The equation code is:

```
EQUATION("NewAgent")
/*
Type of new agent
*/
if(RND<0.5)
 v[0]=1;
else
 v[0]=0;
RESULT(v[0])
```

Computing the type of new agent

The `RND<0.5` is like throwing a random arrow against a target. We know that the arrow can arrive anywhere on the target, with the same probability. We decide that if it hits one half, that we take it as a 0, and if it hits the other half we take it as 1.



Computing the shares

The number of users of MS and Apple increase continually. To better understand how the model proceeds it will be useful to compute the shares for the two types of users.

```
EQUATION("ShareMS")
/* Share of MS users */
v[0]=V("NumMS");
v[1]=V("NumApple");
RESULT(v[0]/(v[0]+v[1]) )
```

and

```
EQUATION("ShareApple")
/* Share of Apple users */
v[0]=V("NumMS");
v[1]=V("NumApple");
RESULT(v[1]/(v[0]+v[1]) )
```

Using smart tricks

The model is simple, but there are quite many elements (i.e. variables and parameters) to be inserted. You can minimize the time requested by using the following shortcuts:

- **Control+d**= Add a descending object. Use the right arrow to move in the Object and press **Enter** to have the Browser showing the new object.
- **Control+v**= Add a variable.
- **Control+p**= Add a parameter.

In all the windows generated when creating elements you type the values requested and press **Enter** to move to the next field.

If you need to change an element, double-click on it in the list, and then double-click on its label in the option window. To remove an element altogether, delete its label.

Defining the model configuration

Let's create the model configuration.

Create an object, call it **Market** and place in it:

1. New agent type: `NewAgent(0)`
2. Utilities for the new agent: `UtilityMS(0)` and `UtilityApple(0)`
3. Number of consumers: `NumMS(1)` and `NumApple(1)`
4. Network externalities coefficients for the two types of agents: `User0Net(P)` and `User1Net(P)`
5. Values for each agent in using each product: `User0MS(P)` and `User0Apple(P)` and `User1MS(P)` and `User1Apple(P)`
6. Share of the number of users: `ShareMS(0)` and `ShareApple(0)`.

Initial values

Let's use the following initial values (menu **Data/Init. Values**):

1. Initial number of consumers is null for both products: `NumMS(1)`=0 and `NumApple(1)`=0
2. Assume that type 0 prefer MS and type 1 prefer Apple: `User0MS(P)`=10, `User0Apple(P)`=5, `User1MS(P)`=5 and `User1Apple(P)`=10
3. Network externalities coefficients are small and positive: `User0Net(P)`=0.2 and `User1Net(P)`=0.2

Set the simulation to run for 1000 time steps (menu **Run/Sim.Setting**), and set the options to **save** `NumMS(1)`, `NumApple(1)`, `ShareMS(0)` and `ShareApple(0)`. Set the option to **Run Time Plot** the **ShareMS** and **ShareApple**.

Expected results

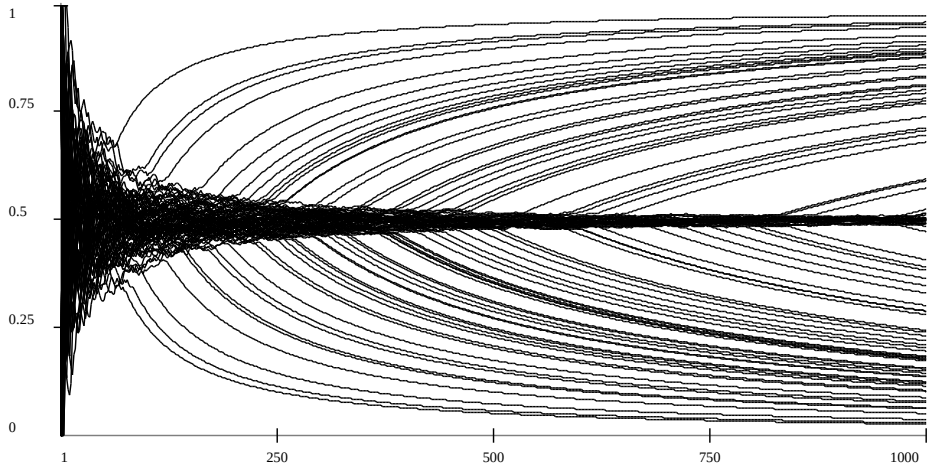
What do you expect from this model? We have an even number of consumers that prefer the two products. That is, each product should take about 50% of the market, if all consumers did choose the product that better satisfies their preferences. But network externalities do change the picture.

How can you test the results of the model with different random series, besides running many simulation runs?

Try to answer, before moving to the next slide.

Results

The model shows that one of the two products wins all the market. However, which product does win is not clear, since changing the seed of the random generator changes also the eventual winner. Instead of running many runs, which is impractical, we can simply multiply the objects **Market** to have many runs in parallel. The graph shows the share of MS for 100 markets.



Expected results

Placing the values of 0.2 we have assumed that the more users exists for a given product the higher is the utility. However, we may test what happens with two other assumptions:

- Negative externalities, the utility *decreases* with the number of existing users. Place -0.2 for **User0Net** and **User1Net**.
- Null externalities, the utility is not affected by the number of existing users. Place 0 for **User0Net** and **User1Net**.

What are the results? Why?