LSD modular models

Let's discuss the interaction between model structure and equations during a simulation run in LSD

The model's equations are generally defined without specific reference on where the values to be used for the computations should be found. It is the system that automatically, at run time, scans the model and identifies the elements to be used, depending on the model structure.

LSD modular models

This feature greatly simplifies the writing of a model, avoiding the possibility of mistakes.

It also adds a large degree of flexibility, as modellers can generally use many different ways to implement the same model, and choosing the most appropriate one for different purposes.

Finally, it makes the models' components (e.g. objects and equations' code) extremely easy to re-use in different contexts, easily allowing recycling of the model's elements.

LSD modular models

In the following exercise we will consider a very simple example, and we will see how the same model code (i.e. the equations) are used by different model structures.

The model we consider is a random walk:

$$\begin{aligned} X_t &= X_{t-1} + RE \\ RE &= U(min, Max) \end{aligned}$$

where $U(m, M)$ is a system-provided (pseudo-)random function providing values as if they were drawn from a uniform function in the range set by the parameters min and Max.

LSD modular models

```
EQUATION("X")
/*
A variable moving as a random walk
*/

v[0]=V("RE");
v[1]=VL("X",1);

v[2]=v[0]+v[1];
RESULT(v[2] )
```

LSD modular models

```
EQUATION("RE")
/*
A random event
*/
v[0]=V("min");
v[1]=V("Max");
v[3]=UNIFORM(v[0],v[1]);
RESULT(v[3] )
```

LSD modular models

Implement the equations in LMM, and then we will create many different model structures that will make use of the same code. The first structure store the two variables and two parameters in an object, say **Obj1** and test the result.

Though a single simulation run appear to generate random results, repeating the simulation with the same setting generates always the same identical series.

LSD uses the standard system of pseudo-random number. Changing the *seed* of the simulation provides completely different values.

Before running a simulation, open *Sim.settings* in menu *Run* to change the seed.

LSD modular models

Programming languages cannot produce genuine random values, but use *pseudo-random number generators*. These are (deterministic) functions that provide sequences of values whose statistical properties are identical to a random series, such as uniform, normal, gamma, etc.

The *seed* is a sort of index cataloguing the random series. If you use the same seed the same series is used, and the same results can be consequently reproduced. Changing the seed makes the generator produce a totally different series.

LSD modular models

The second model structure we consider is the same model replicated in many different copies

$$\begin{aligned} X_t^i &= X_{t-1}^i + RE_t^i \\ RE_t^i &= U(min^i, Max^i) \end{aligned}$$

To generate this model the user has simply to multiply the object **Obj1** in the model from 1 to many copies, and the automatic data retriever will ensure that the i^{th} copy of X will make use of the i^{th} copy of RE. This is ensured by the fact that LSD will use of elements within the same objects of the variable under computation.

LSD modular models

A third possible structure is based on the consideration that all the copies of parameters min and Max have the same value, and it is therefore useless to have many copies. Consider the model equivalent to the following notation:

$$\begin{aligned} X_t^i &= X_{t-1}^i + RE_t^i \\ RE_t^i &= U(min, Max) \end{aligned}$$

LSD modular models

The difference with the previous models is that while X and RE are assigned an index, the parameters min and Max are not, implying that they are unique.

To implement this model generate a new structure, made of object **Obj1** containing another object type, **Obj2**. Place min and Max in **Obj1** and X and RE in **Obj2**. Finally, generate many copies (say 10) of **Obj2**. Running the model you will see that LSD is able to re-use the existing equations with the new structure.

LSD modular models

In this model structure when the system tries to compute a RE it cannot find the required elements min and Max within the same object containing RE. The data retrieving rules state that, if you can't find the required elements in the same object, then tries in lower level, descending objects and. Failing this (as in our case), look "up" in the parent object. Consequently the "centralized" parameters will be found and used by all different copies of RE.

LSD modular models

Another complication emerges when we generate a "deeper" model structure, where objects at different levels are replicated in many copies. Therefore, the previous model becomes:

$$\begin{aligned} X_t^{j,i} &= X_{t-1}^{j,i} + RE_t^{j,i} \\ RE_t^{j,i} &= U(min^j, Max^j) \end{aligned}$$

To implement this, make several copies (2 are sufficient) of **Obj1**. To differentiate the model assign the values of $min = -1$ and $Max = 1$ to the parameters in the second object. In this way it will be evident whether the X result from the first or the second copy of **Obj1**.

LSD modular models

The model will, again, behave as we expect. Every RE descending from the j^{th} from the same "high level" object will automatically make use of the "correct" parameters. This is ensured by the fact that the data retrieving starts the search from the very object containing the variable computed, and therefore will scan its "branch" of the model structure. To confirm this, let's observe the model while running a simulation with the LSD debugger.

Before running the simulation, check on the options **Debug** for variables X and RE, and set in *Sim.setting* the value for **Insert debug at** to 1. This will cause LSD to interrupt the simulation when one of the variables marked as debugged is computed, showing the content of the model at that moment.

LSD modular models

Let's consider now RE. This is a variable that must obviously be newly computed for each X in the model, to avoid that the same random value is used by different X's. However, to have several copies of it is a waste, since we do not need to store their values, after they have been used in X. The following model can also be implemented in LSD, provided we pay attention to the nature of RE.

$$\begin{aligned} X_t^{j,i} &= X_{t-1}^{j,i} + RE_t^j \\ RE_t^j &= U(min^j, Max^j) \end{aligned}$$

LSD modular models

The model contains a single RE for all the X contained in a branch of the model. If we declare RE as a variable, than it would be a mistake. This is because a variable is computed only once at each time step. Therefore, the same value RE_t computed at one time step would be applied to all the X's.

Instead, what we need is to have the equation of RE to be re-computed every time it is requested in the equations for a X, recomputing its code even several times within the same time step. To accomplish this, we must declare RE as a *function*, and not as a variable.

LSD modular models

To change the nature of an element, or to change its position, double-click on its label in the Model Browser, and then, again, on its label in the options' window. The resulting windows allows to move the element to a new object; to change its label; to delete it (assign an empty label); to change its nature.

Use this window to move *RE* from **Obj2** to **Obj1**. Then, re-use it again to turn *RE* into a function.