

General Guide to Design a NSL-MatLab Model

University of Southern California

A supplement to Prof. Arbib's CS 564, Brain Theory and Artificial Intelligence

University of Southern California, Fall 2005.

Preface

For now, it is the time to introduce how to design a NSL-MatLab model. The basic principles and technique which are used in the NSL-MatLab is the main concern of this. We provide the whole view between NSL and MatLab in the Nsl-MatLab tutorial. Though that shows insight of how to design a model, this is too complex for people who does not know the previous NSL-J or NSL-C to apprehend. On the contrary, this document is different in aspect of a range of view, Independent to the previous NSL structure, it is focused how to organize an overall structure and how to construct the computational part of a model.

It is strongly notice that this draft can be modified to apply to other complex model in future. If you are facing the problem with this standard while constructing a model because this standard is not sufficient to support your idea, please let us know.

0. Overall procedure to design a model

Here is the summarized procedure.

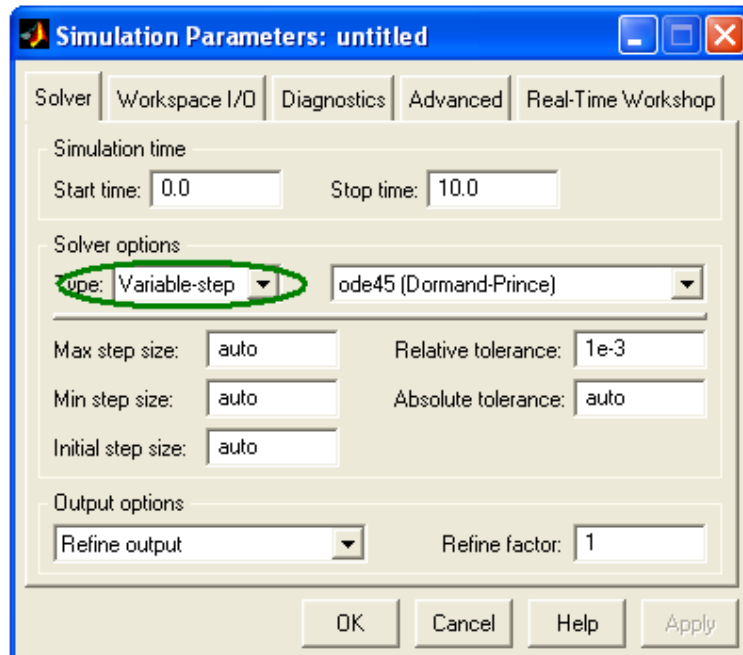
- A. Decide whether the system is time-based or sample-based
- B. Distinguish between memorized variables(memory or state), non-memorized variables(potential), and parameters
- C. Organize the hierarchical structure of a model
- D. Design a module: non-leaf and leaf module
- E. How to write dimension-free computational part

1. Decide whether the system is time-based or sample-based

One of most important idea is whether the system is time-based or sample-based; this is a different question, whether the system is continuous or discrete. Difference between time-based and sample-based lies on the type of input data set. As an example, if the input is time variant matrix or vector like MaxSelectorModel, the system is time-based. On the contrary, in case of

BackPropModel, because the input vector is a discrete set of training data, whose each row does not mean the time index. Even though system is sample-based, overall computation is usually continuous.

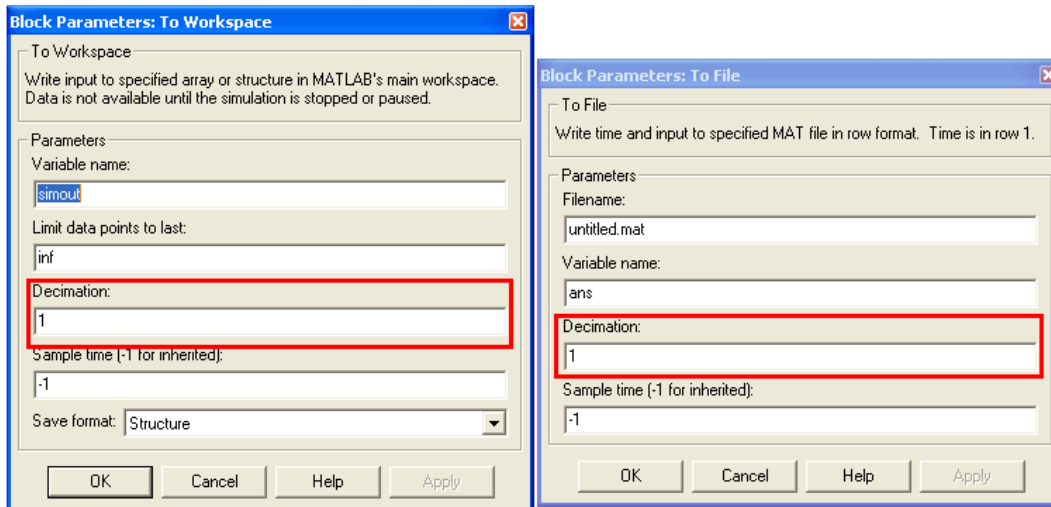
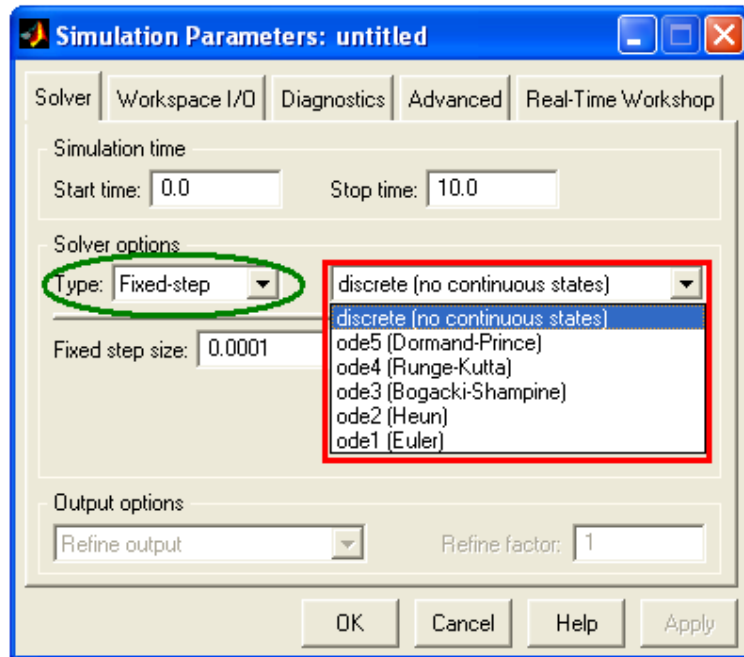
In case of time-based model, everything is fine; let it go. A noticeable point is that the step size field in the simulation parameter window below. The step size is the size between the each point. Even though the simulation is continuous, because the computer is not continuous machine, the time should be discretized. However, if the step size is enough small, the system is little different from the continuous system. Furthermore, MatLab usually set the step size wisely. In the region where the variable varies small, it changes the step size also small. As doing so, MatLab obtains better accuracy efficiently.



If it is necessary to get a value of the system every some seconds, you should change the step size to “fixed-step” and set the step size enough small. First, it is because “to workspace” and “to file” block supports saving every some decimation but every some time. That decimation option in both parameter windows of “to workspace” and “to file” decides how often saved the data. If decimation is set with 10, the data will be saved every 10 step. Here is a practical example. If the fixed step size is 10^{-6} and you want to record the data every millisecond, decimation option should 1000. Of course, setting the step size is up to the model.

In this case, another solver option should not be chosen with “discrete”; it does not handle continuous system. However, other “ode-” options enable the system is continuous. As some of you know, the previous NSL provides ode4(Runge-Kutta) and ode1(Euler) to solve the NslDiff. Notice that too small step size with ode5 is much slower than the variable step size with ode45.

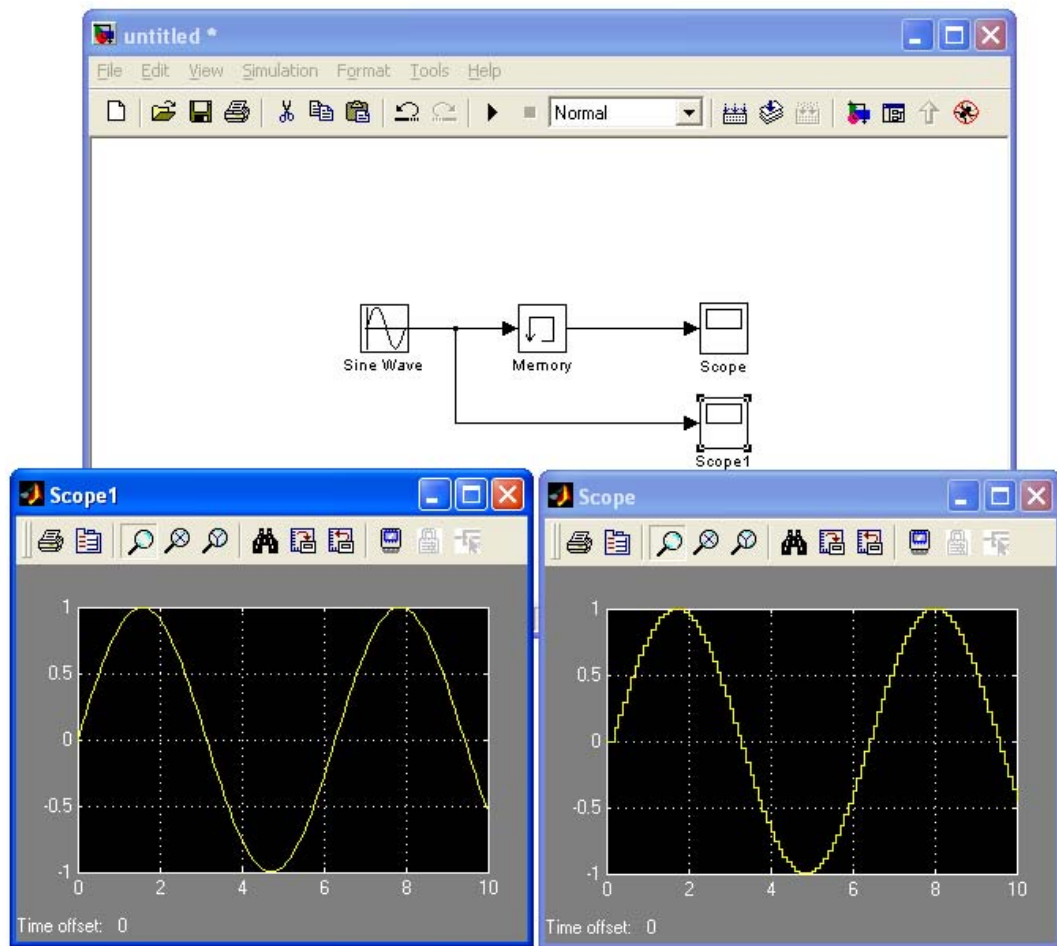
Setting the step size and stop simulation time is a issue of time saving. I recommended that before change the parameter with fixed step size, check the simulation end time using variable step size. Then, saving the valuable experimental data, set the simulation end time at least.



In case of sample-based model, the setting dose not look so different. However, this is completely different because the special block controls the system. The main idea is matching the each row of data into the each step of time. Outside the model, data sample pusher is required. This special block’s main blocks are “Memory Block” and “Select Block”.

The pusher has the “Memory” block at the very before the out port. The memory block keeps the current value until step reaches the next. As an example, in the below diagram, if the step size is 0.1,

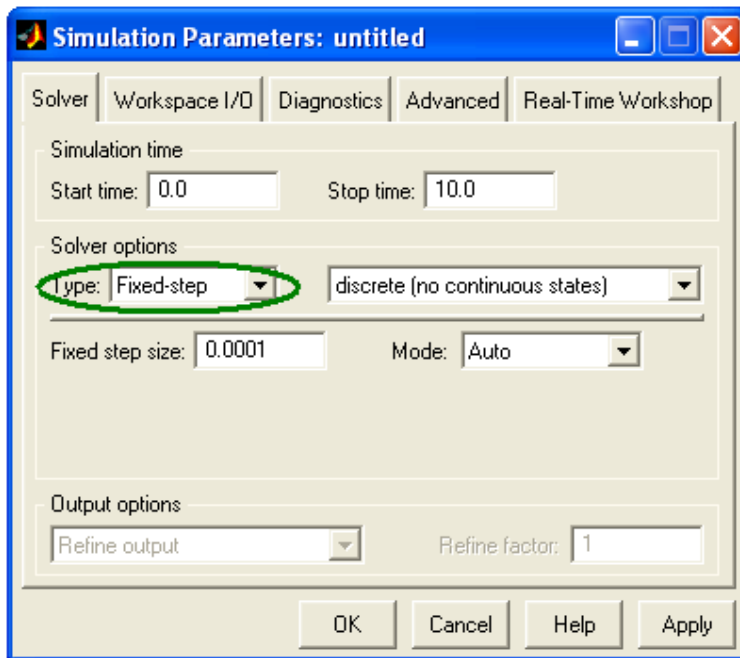
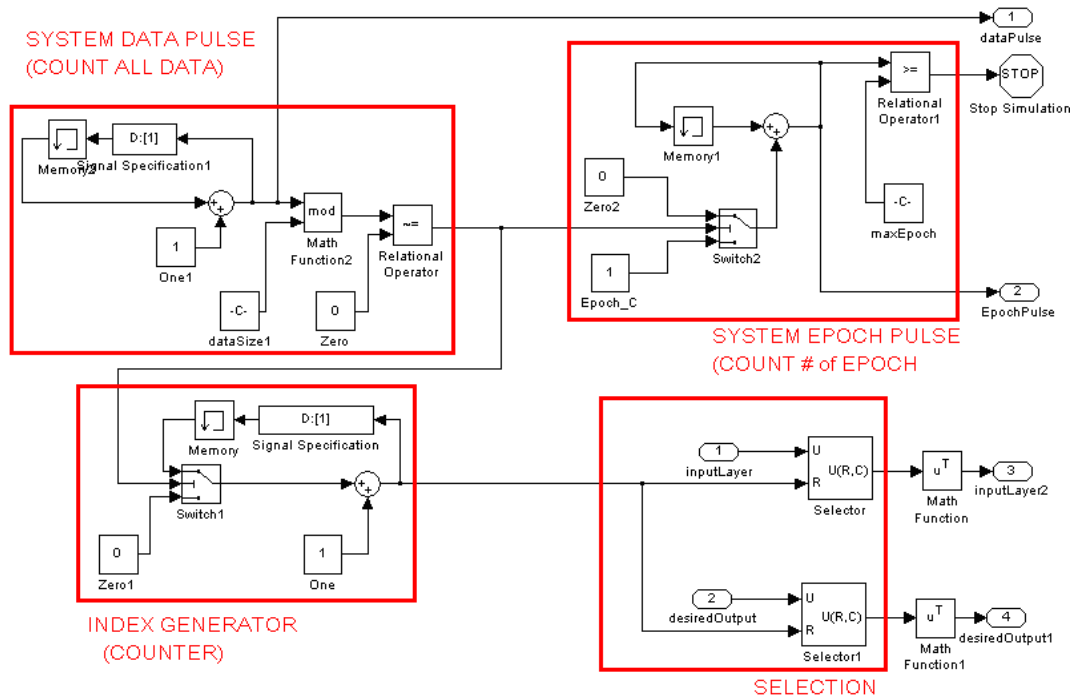
the output after memory is discretized because the memory block keeps the value for the step size.



“Select Block” enables the pusher to put the specific one row into an out port. Also using the memory block, a counter can be constructed and it will be the index to select. Below is the pusher of BackPropModel.

This Pusher has another advantage, the system pulse. This system pulse indicates how much the system has run. As an example, the epoch pulse in the BackPropModel is issued whenever the data set is learned completely once. Thus, using this signal, the error function can be reset.

This technique, mapping the data set into the time space, is similar to a technique of VLSI design. They use a clock signal to synchronize all memory and computation. In our MatLab environment, clock signal is hidden. However, the concept, that memory is updated every some step, is same. Comparing with using explicit clock generator to synchronization, this technique is much simpler because the clock signal does not have to be connected with every memory block as like in the VLSI design.



Surely, time-based and sample-based can be mixed. It is simply obtained with setting “fixed-step” and “ode-” in the simulation parameter window and adding the specially designed pusher. Notice that this pusher should keep the out value for a while to allow the time-based computation enough. This keeping the value can be implemented with a counter structure.

2. Distinguish between memorized variables(memory or state), non-memorized variables(potential), and parameters

Another big issue is the type of variable. This is decided by dependency to the input value; whether the value is directly related with the input value or not. There are three different types and distinction is very important: memorized variables (memory or state), non-memorized variables (potential), and parameters.

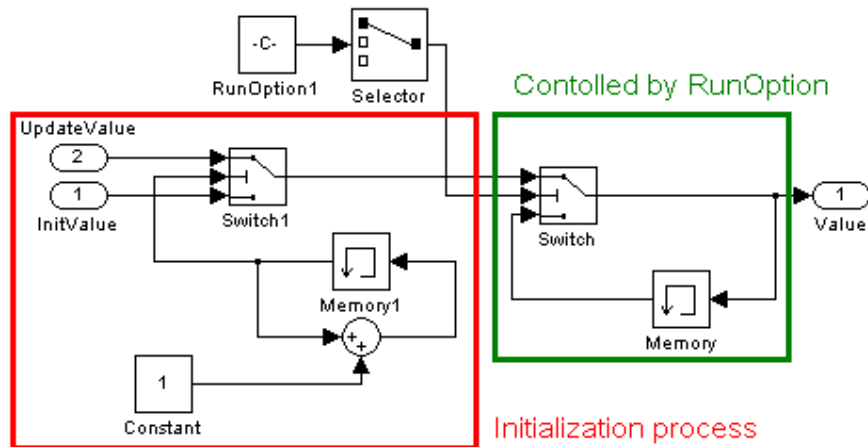
First, some values are independent from input values. Even though the initial value is inputted at the first time, the value is not the one which is passed by the other block. Instead, this value is kept or trained in the block after instantiate once. This kind of variable can be considered as memorized variables. As an example, the weight matrix and threshold vector are memorized variables. In other words, it can be called as “state” because the value is dependent on the current state.

Second, some values are directly or indirectly dependant from input values. In this case, because the value can be derived from input data and other memorized variables, it does not have to be memorized. The example of this kind of value is a potential of MaxSelectorModel’s neuron mf or a potential of BackPropModel’s neuron mf. So, temporarily it will be called as “potential”; we are finding a better name for this.

Last type of value is very different from two above. A user who simulate his model try to obtain the experimental data by setting various parameter. This kind of information may change the model itself a little without touching overall hierarchical structure, as an example, the size of neural network. There is another advantage of NSL-MatLab, to set this parameter without re-compiling.

Implementation techniques of these values are briefly introduced below.

A memorized variable is implemented as “TimeUpdateVariable” in the Important math section. Of course, it is simply implemented with “memory block.” However, to enable setting the initial value explicit and to be controlled with RunOption, below structure is implemented. One point is the position of the memory block in the green box. As setting like these, it delivers the initial value at the very first step.



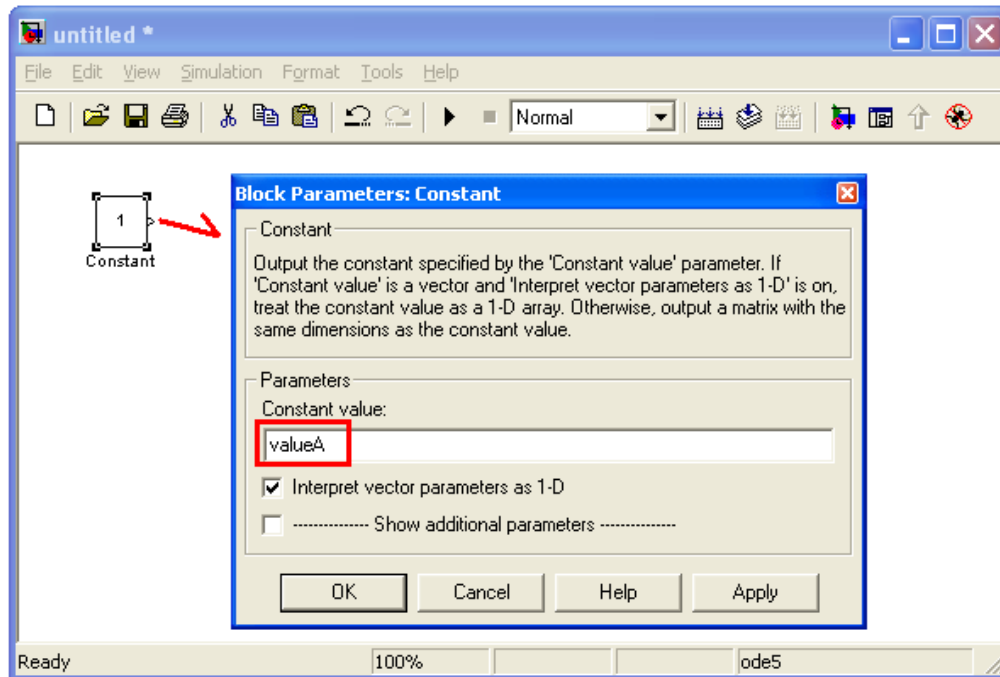
Model Info

Created by Han, Cheol
 This is final dimension-free version
 The final value is selected by RunningOption,
 which is initialized in "NsInit"
 Cf)
 RunOption is 3by1 vector. Here's meaning of them.
 RunOption(1): training;1 otherwise:0
 RunOption(2): running;1 otherwise:0
 RunOption(3): hold;1, otherwise:0s

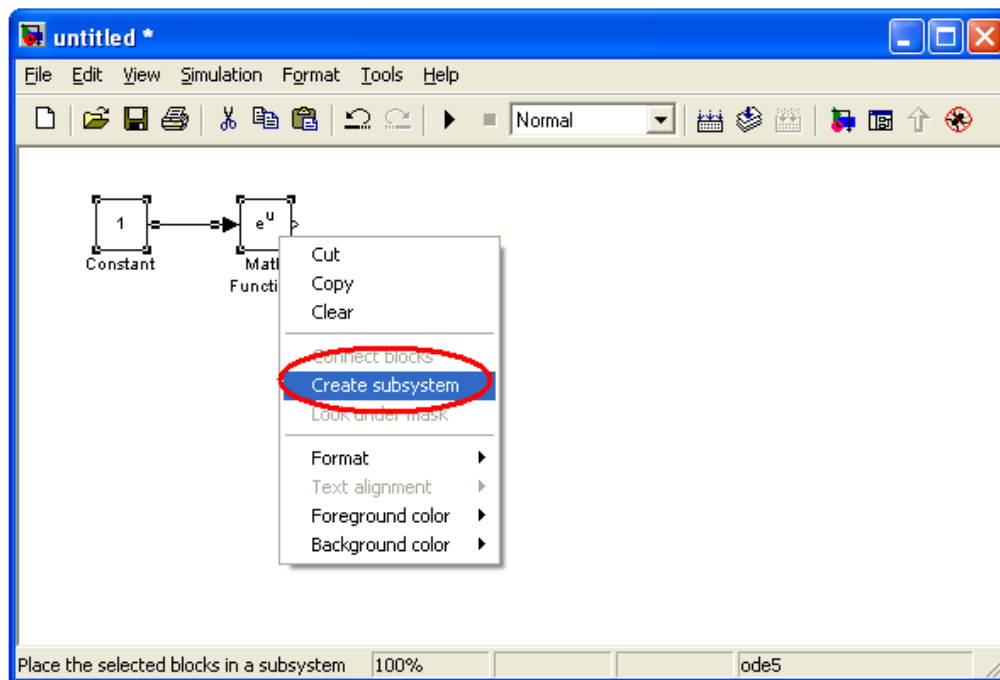
Non-memorized variable is any signal flow in the diagram. It is not indicated but can be named by double-clicking the signal line. To save this signal, it is recommended to use “to workspace”, “to file” or “to file with time index”; refer the related section in the NSL-MatLab tutorial.

To make parameter free to setting is not so difficult but more complex than above two. It is a recommendation to understand the section “How to utilize the data in the workspace” before reading this. The example exist also in the above “TimeUpdateVariable.” RunOption is parameter and it is maintained in the constant block. As changing the data in the workspace, the parameter in the model is also changed. One another method is using SubSystem property of MatLab; this is more plausible in order to expand our NSL library under the concept of modularity. While constructing model, the parameter should be contained in the constant block. Difference is that the field of the constant block is filled with any name, whether it is in the workspace or not. Then create the subsystem including that constant block. Last, mask the subsystem and put the parameter. Below the process is explained in the figure. Surely, you can add the parameters as many as you want.

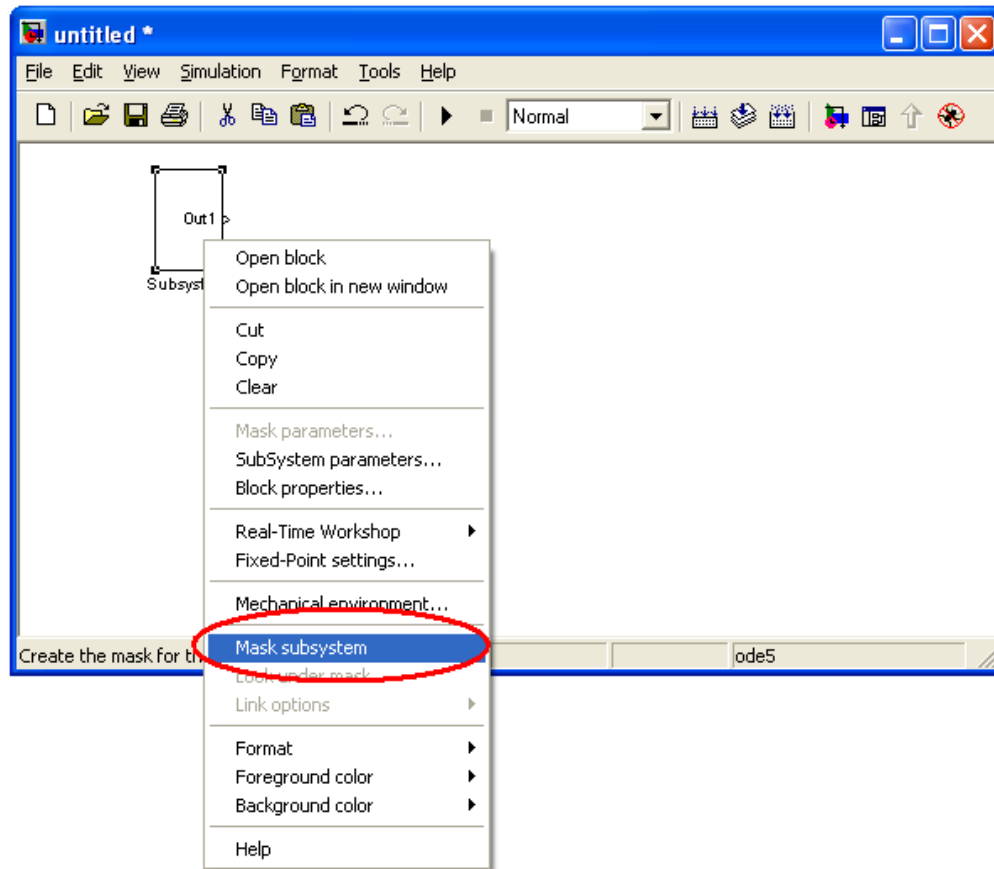
1. CREATE a CONSTANT BLOCK and FILL a NAME in ITS PARAMETER



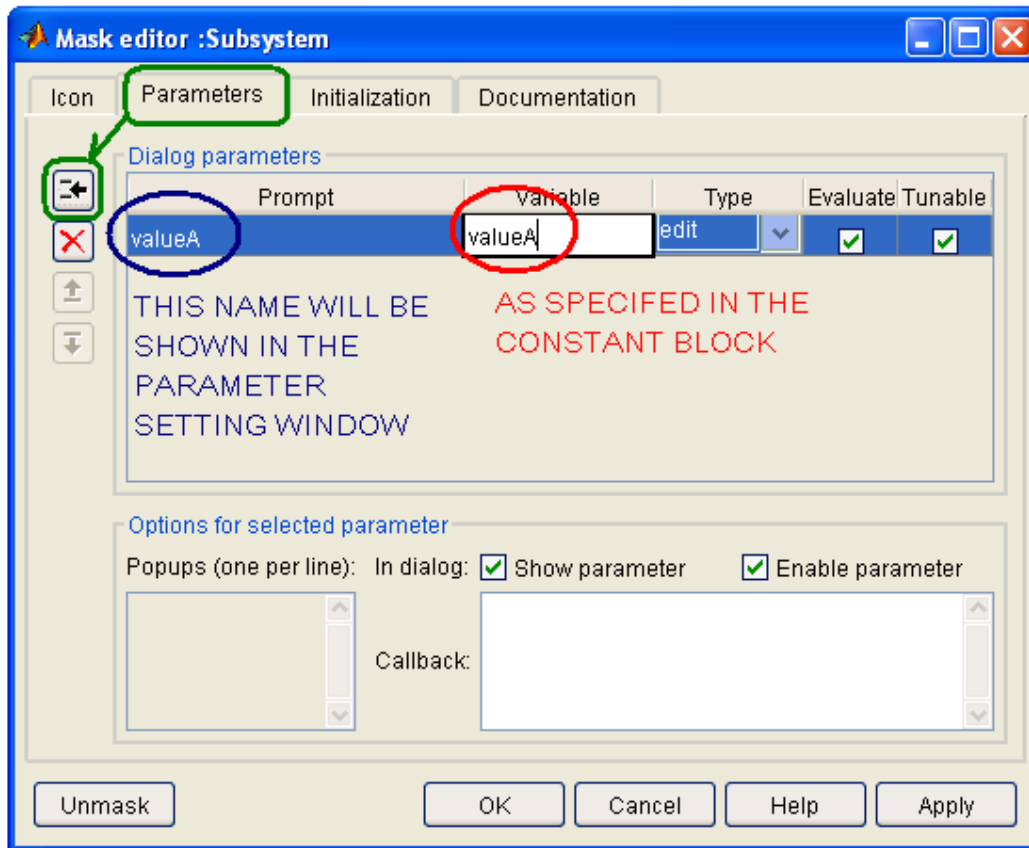
2. SELECT WHAT YOU WANT TO MAKE AS A SUBSYSTEM
And RIGHT CLICK on those, then selectr "CREATE SUBSYSTEM"



3. SELECT "MASK SUBSYSTEM" AS RIGHT CLICK ON SUBSYSTEM BLOCK



4. PUT THE PARAMETER



3. Organize the hierarchical structure of a model

It is an advantage that NSL-MatLab shows the hierarchical structure of model transparently. Thus, we should pay more attention on organizing the overall structure. Of course, even if this structure is ignored, the simulation shows the correct result. However, that's what we want. Thus we recommended the following rules at least. **NEED MORE?**

A. Try to follow the biological structure

If the brain region A is revealed that it is included in the specific brain region B, the module A should be capsulized in the module B conceptually even though the module B does not have anything except brain region A. In this case, the module B is a non-leaf module. And if the module A has only computation part and does not have any other module which does not represent a brain region, the module A is called as a leaf module. Usually, non-leaf module does not have computational part but have many non-leaf and leaf module. On the contrary, leaf module does not have anything except computational part.

Furthermore, the same level of diagram must show the same level of brain concept; as an example, visual cortex versus motor cortex or v1 versus v5.

B. Try to indicate the functional region

In the same brain region, functional difference should be implemented separately to modularization.

C. Try to make pieces as many as you can

It is a virtue of computer scientists.

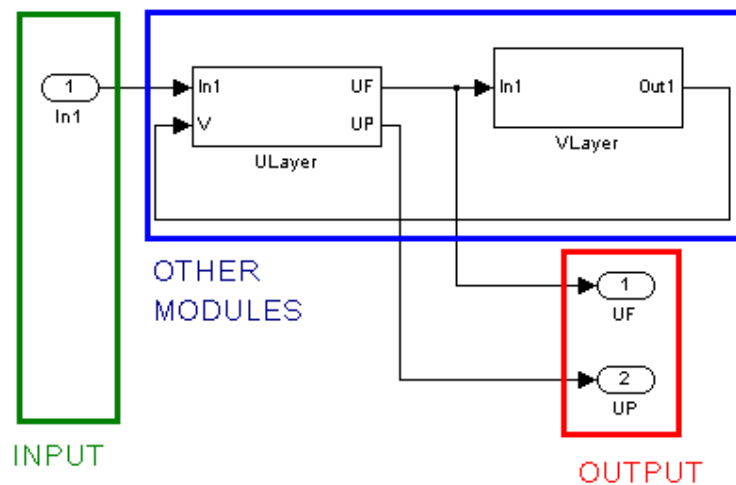
4. Design a module: non-leaf and leaf module

While constructing a diagram of each subsystem, there is also a convention. Traditionally, a main signal flow is from left to right. Following this tradition gives feeling that it is arranged properly. It does not matter whether the signal flow is from-top-to-down or from-down-to-top; however, from-top-to-down is preferred.

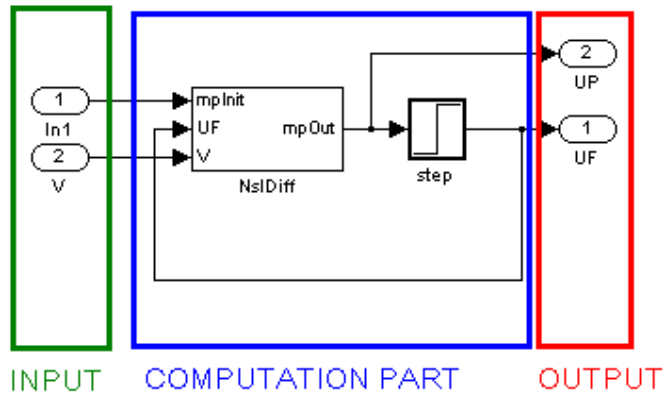
Another big and hard premise is that arrow means that there is not a counter flow. If it is necessary to express that a counter flow is possible, it is recommended to add one more arrow with the opposite direction.

As following this convention, to place in-ports, variables and parameters in the leftmost is preferred. On the contrary, to place out-port in the rightmost is also better; but not so strict. It can be violated if the flow line is so complex for doing so. Of course, a non-leaf module's middlemost has any other non-leaf modules and a leaf module's has computational part. Here is an example in the MaxSelectorModel.

NON-LEAF MODULE : MaxSelector



LEAF MODULE: U-LAYER



5. How to write dimension-free computational part

What is dimension-free? As mentioned above, some parameters can be changed to fulfill the experimental purpose. However, whenever the size of neural network is modified, should we change the computation part? It is expected that that kind of pains does not occur. So, when constructing a model, a modeler should be careful to whether the structure is free to the size. Moreover, the desire to make it free to dimension exists. As an example, FeedForward module basically handles a 2-dimensional weight matrix and a 1-dimensional input vector. However, a little more care helps to make it can handle 3-dimension and more. This is dimension-free.

This can be obtained easily because almost all the blocks in the basic simulink library are dimension-free. **Do not use FCN block**; it cannot handle even 2-dimensional matrix. Additionally, S-function which will be introduced in the next section should be avoided if the technique which is used in the library currently is not applied.

6. How to speed up computation by vectorizing

7. S-function: .m file for dimension-free computation

S-function is the simulink block to enable .m file or c file to run in the simulink simulation. Thus, it is one of crucial part of MatLab. However, explaining whole of this is not easy. So, currently it is omitted; however, people needs a lot, it will be provided.

Because .m file can handle only 1 dimensional vector, in order to handle matrix input MEX(c code) file is required. However, in our library, using a tricky method to deliver the dimension information into the .m file s-function, MEX file is not required. So, instead using MEX, which is much faster than .m file but complex, .m file is used for dimension-free s-function in our

library. It is not as complex as MEX does. If you need it and are ready to learn it, please post the message on the board. We will prepare for it.