Tutorial for LabVIEW Simulation Module

By Finn Haugen (finn@techteach.no), TechTeach
October 8 2004

This document is freeware – available from http://techteach.no

Contents

1 The Simulation functions palette ........................................ 3
2 An expressive example: A PID control system for a nonlin-
ear process model .................................................................. 6
3 Various topics ........................................................................ 12
  3.1 How the Simulation module works .................................. 12
  3.2 Representing state space models using integrators and For-
mula node ..................................................................................... 13
  3.3 Using program flow structures in a simulation diagram .... 13
  3.4 Creating subsystems ............................................................. 13
  3.5 Getting a linearized model of a subsystem .................... 14
  3.6 Exchanging data between simulation loops and while loops 15
  3.7 Importing models from Control Design Toolkit and System
    Identification Toolkit ................................................................. 16
    3.7.1 Importing models from Control Design Toolkit ........ 16
  3.8 Translating SIMULINK models into LabVIEW Simulation
    models ....................................................................................... 17
Preface

LabVIEW Simulation Module\textsuperscript{1} is a block diagram based environment for simulation of linear and nonlinear continuous-time and discrete-time dynamic systems. Many simulation algorithms (i.e. numerical methods for solving the underlying differential equations) are available, e.g. various Runge-Kutta methods.

The mathematical model to be simulated must be represented in a simulation loop, which in many ways is similar to the ordinary while loop in LabVIEW. You can make the simulation run as fast as the computer allows, or you can make it run with a real or scaled time axis, thus simulating real-time behaviour, with the possibility of the user to interact with the simulated process. The simulation loop can run in parallel with while loops within the same VI.

The files used in the examples in this document are available from the home page of this document at http://techteach.no.

If you have comments or suggestions for this document please send them via e-mail to finn@techteach.no.

The document may be updated any time. The date (written on the front page) identifies the version. Changes from earlier versions will be commented here in this preface.

Version dated October 10 2004 vs. August 29 2004: The functions of the Simulation Palette are now listed in the beginning of the document, and the pictures of the subpalettes at the end of the document have been removed. The document contains a new figure (Figure 6).

\textit{Finn Haugen}

\textsuperscript{1}Produced by National Instruments
1 The Simulation functions palette

Once the Simulation Module is installed, the Simulation palette is available from the Functions palette. The Simulation palette is shown in Figure 1.

![Simulation Palette](image.png)

**Figure 1: The Simulation Palette**

Below is an overview over the functions and possible subpalettes on the Simulation Palette:

- **Simulation Loop**, which defines the borders of the simulation diagram on which the simulation functions are placed. Note that (most of) the functions found on the Simulation palette can be placed only inside a simulation loop.

- The **Continuous** palette, with the following functions and/or subpalettes:
  - **Zero-Pole-Gain Function Space Delay**
  - **Derivative**
  - **Time Delay**
  - **State Space**
  - **Transfer Function**
  - **Zero-Pole-Gain**
• The **Nonlinear** palette, with the following functions:
  
  – *Backlash*
  – *Friction*
  – *Quantizer*
  – *Dead Zone*
  – *Rate Limiter*
  – *Relay*
  – *Saturation*
  – *Switch*
  – *Zero Cross Detection*

• The **Discrete** palette, with the following functions:
  
  – *Discrete Integrator*
  – *Discrete Unit Delay*
  – *Discrete Zero-Order Hold*
  – *Discrete First-Order-Hold*
  – *Discrete State Space*
  – *Discrete Transfer Function*
  – *Discrete Zero-Pole-Gain*
  – *Discrete Filter*

• The **Signal Generation** palette, with the following functions:
  
  – *Chirp Signal*
  – *Ramp*
  – *Pulse*
  – *Signal Generator*
  – *Sine Wave*
  – *Step*

• The **Signal Arithmetic** palette, with the following functions:
  
  – *Gain*
  – *Summation*, with any number of inputs. Inputs may also be subtracted.
  – *Multiplication*, with any number of inputs. Inputs can slo be used as divisors.
- The **Lookup Tables** palette, with the following functions:
  - *Lookup Table 1D*
  - *Lookup Table 2D*
  - *Lookup Table 3D*

- The **Utilities** palette, with the following functions:
  - *Collector*, which collects a signal at each time step of the simulation and outputs a history of the signal value and the time at which this function recorded each value in the history. The Collector builds an array on its output from the input it receives at each simulation step.
  - *Indexer*, which is used to index an array or waveform (like the ordinary array indexing function in LabVIEW) by the current simulation time.
  - *Simulation Parameters*, which can be used to read or get (not set) simulation parameters.
  - *Simulation Time*, whose output is the simulation time. It works like a simulation clock.
  - *Report Sim Error*
  - *Halt Simulation*, which stops the simulation when its input becomes False.
  - *Set Diagram Params*, which may be used to programmatically set simulation parameters, and it must then be connected to the parameters terminal up left on the simulation loop.

- The **Graph Utilities** palette, with the following functions:
  - *SimTime Waveform*, which is used to plot simulated variables continuously in a LabVIEW Chart. When this function is put into the simulation loop, a Chart is automatically attached to the function.
  - *Buffer XY Graph*, which is used to collect arrays of signals to be plotted in a LabVIEW Graph. When this function is put into the simulation loop, a Graph is automatically attached to the function.
2 An expressive example: A PID control system for a nonlinear process model

As an expressive example, let us consider the simulation of a PID control system for a process model having saturation on its input – thus, the process model is nonlinear.² Figure 2 shows the front panel of the simulator. On the front panel is shown a block diagram of the control system. The linear part of the process model is given by the transfer function

\[ H(s) = \frac{3}{1 + 2s} \]  

²It is assumed that you have basic knowledge about control theory [2], but if you don’t, you should still be able to follow the example.
The saturation limits are

\[ u_{\text{max}} = 5; \quad u_{\text{min}} = 0 \]  \hspace{1cm} (2)

The PID controller settings are

\[ K_c = 1; \quad T_i = 0.02 \text{min}; \quad T_d = 0 \]  \hspace{1cm} (3)

Figure 3 shows how the block diagram drawn on the front panel shown in Figure 2 can be represented in a simulation diagram or a simulation loop in LabVIEW.

Below are comments to the simulation diagram shown in Figure 3.

- The **Transfer function block**: It represents the transfer function (1): The block is copied into the simulation diagram from the Continuous palette. The block can be configured by double-clicking its block, causing a configuration dialog window to be opened, see Figure 4. In the **Parameter source** field in that dialog window you can select between **Configuration page** and **Terminal**. By selecting **Configuration page** you must define the numerator and denominator parameters of the transfer function on the configuration page (or the dialog window) itself. By selecting **Terminal** an input terminal is
Figure 4: Configuration window for the transfer function in the simulation diagram shown in Figure 3.

created on the left part of the block, and you must connect a cluster of two arrays whose elements are the parameters. The numerator and denominator parameters are given as arrays whose elements are the coefficients of *ascending* orders of *s* (or *z* for a discrete-time transfer function). Thus, for the transfer function (1) the array representing the numerator polynomial is

\[ \begin{align*}
3 & \\
\end{align*} \] (4)

and the array representing the denominator polynomial is

\[ \begin{align*}
1 & \\
2 & \\
\end{align*} \] (5)

Although it is not used in the present example, you may flip a block right-clicking the block and selecting the menu **Reverse terminals**.

- **The Summation block**: The block copied from the Signal Arithmetic palette. The block can be configured by double-clicking the block, thereby opening a configuration window. You can then add inputs, select signs (+ or −), or remove inputs.

- **The Saturation block** is copied from the Nonlinear palette. The block can be configured by double-clicking the block, thereby
opening a configuration window. In the Parameter source field in that window, I have selected Terminal to create one block input to define the maximum value of the saturation block output, and another block input to define the minimum value of the saturation block output, see Figures 3 and 2.

- **The PID.vi block** is copied from the Control palette (the PID subpalette) which is not a part of the Simulation Module. This is an example of using a block or function in a simulation diagram which is not on an originally a simulation block (it is not found on the Simulation palette).

  Note that the PID block has got a red D on its icon (this happens to other ordinary LabVIEW functions, too). The D stands for Discrete. It means that the function (the block) is called as a discrete-time block once per simulation time step. By right-clicking on the block icon and selecting the menu SubVI Node Setup you can define a different Rate divisor than the default value of 1, e.g. 2 which causes the function to be called each second time step.

  Note also that the sampling time of the PID controller is set equal to the simulation step time (0.01s).

- **The SimTime Waveform block** is copied from the Graph Utilities palette. The block is used to prepare signals to being plotted in a Waveform Chart having automatically a correct time scale (equal to the simulation time scale). When you insert a SimTime Waveform block into a simulation diagram, a Waveform Chart is automatically inserted on the front panel.

- **The Build Array block** is copied from the ordinary Array palette. It is used to collect the two signals to be plotted, namely the setpoint \( y_{SP} \) and the process output variable \( y \).

- **Setting simulation parameters**: Simulation parameters as solver method (i.e. method of numerical solution of the differential equations constituting the simulated model), time step, and initial time and final time, may be set by right-clicking the simulation loop border and selecting the menu Configure Simulation Parameters. This opens the Simulation Parameters dialog window, see Figure 5.

  If you want the simulation to run until the user clicks a Stop (Halt) button on the front panel, as in Figure 2, you can set Final Time to Inf (infinity).

  Regarding continuous solver method, it is my experience that using the Runge-Kutta second order method (with a fixed time step) works
Figure 5: The Simulation Parameters dialog window, which is opened by right-clicking the simulation loop border and selecting the menu Configure Simulation Parameters.

fine. The simulation runs accurately without unnecessary time delay which otherwise may occur due to numerical complications.

Usually I find the time step using the following simple procedure: If the smallest time constant of the model is $T_{\text{min}}$, I set the time step $h$ to

$$h = T_{\text{min}}$$  \hspace{1cm} (6)

If I do not know the smallest time constant, I set $h = 0.1$ and reduce it until it does not influence the accuracy of the simulation.

In Figure 5 the Discrete Time Step Multiple parameter is the number of times discrete-time functions (blocks) are called at each time step. The default value is 1.

- **Setting simulation parameters programmatically:** As an alternative to setting the simulation parameters via the Simulation Parameters dialog window as explained above, you can set the parameters programmatically by connecting a Set Diagram Parameters function, which is found on the Utilities palette, to the simulation parameters terminal on the upper left corner of the simulation loop, see Figure 6. The input to the Set Diagram
Simulation parameters can be set programmatically by connecting a Set Diagram Parameters function (on the Utilities palette) to the simulation parameters terminal on the simulation loop.

Parameters function is a cluster of simulation parameters, see Figure 7.

Figure 6: Simulation parameters can be set programmatically by connecting a Set Diagram Parameters function (on the Utilities palette) to the simulation parameters terminal on the simulation loop.

Figure 7: Simulation parameters on the front panel of the simulator

- **Setting simulation timing parameters:** By right-clicking the simulation loop border and selecting the menu Configure Simulation Timing Parameters, the Loop Configuration dialog window is opened, see Figure 8. The Period parameter specifies the amount of time between two subsequent simulation loop iterations. By setting the Period parameter equal to the simulation step time the simulation runs in *real time*. By giving the Period parameter some other value, the simulation time scale is proportional to real time. For example, if
the simulation time step is 0.01s, setting Period equal to 0.05 causes the simulation to run 5 times slower than real time. Setting Period to 0 runs the simulation as fast possible (on the computer).

3 Various topics

3.1 How the Simulation module works

Immediately before a simulation is started, LabVIEW transforms the block diagram model to a model consisting of differential equations and/or differential equations. Then the selected numerical solver method is applied to calculate the responses. The model transformation takes some time. Therefore the simulation does not start immediately after the Run button is pressed, but after some time depending on the complexity of the model.
3.2 Representing state space models using integrators and Formula node

Using integrators from the Continuous palette and the Formula node from the general Analyze/Mathematics palette, nonlinear state space models may be expressed compactly. Using the Formula node you can avoid drawing the model using many single blocks, as the adder and the multiplier on the Signal Arithmetic palette.

Let us look at a simple example of using the Formula node. Given the following state space model:

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -x_1 + u \\
y &= x_1
\end{align*}
\]

(which is a state space model of an oscillator). \(u\) is the input variable, and \(y\) is the output variable. Figure 9 shows the front panel and the simulation diagram of a simulator for this system. Note how a Formula node is used to represent the right side of the differential equations (7)–(9). The integration of the time-derivatives are performed by Integrator blocks from the Continuous palette.

3.3 Using program flow structures in a simulation diagram

You can use a majority of LabVIEW VIs and functions to describe a model. However, you cannot place certain structures, such as the Case structure, While Loop, For Loop, Event structure, or the Sequence structures, directly on the simulation diagram. Instead, you can place these structures in a subVI and then place the subVI on a simulation diagram.

3.4 Creating subsystems

You can create a subsystem of a part of a simulation diagram in much the same way as you create ordinary subVIs from parts of an ordinary diagram. Let us create a subsystem of the Formula node and the integrators in the simulation diagram shown in Figure 9. The first step is to select or mark the part of interest. Then the subsystem is created using the menu Editor / Create Simulation Subsystem. Figure 10 shows the resulting main simulation block diagram, and Figure 11 shows the front panel and the diagram of the created subsystem.
3.5 Getting a linearized model of a subsystem

LabVIEW can create a linear state space model from a linear or nonlinear subsystem. (Creating subsystems is described in Section 3.4.) The procedure is to select or mark the subsystem of interest, and then create the linear model by using the following menu: Tools / Simulation Tools / Linearize Subsystems. You are given the option of saving the linear model as a model (to be used by functions in the Control Design Toolkit) or as a VI containing the state space model in the form of a cluster of coefficient arrays. Perhaps the most flexible choice is VI. Figure 12 shows as an example the state space model resulting from linearizing the subsystem shown in Figure 11.\(^3\)

\(^3\)The resulting model is in this case identical to the original model – except for the ordering of the states, as it should be since the original model is linear.
3.6 Exchanging data between simulation loops and while loops

It is in principle possible to put (almost) any LabVIEW code for e.g. analysis and design into a simulation diagram, but doing so may cause the simulated model become unnecessarily large, which may slow down the simulation time. Because of this delay, you should not put more code inside the Simulation Loop than is strictly necessary for representing the model to be simulated. But where to put (analysis or design) code? In one or more ordinary While loops running in parallel with the Simulation Loop. Data can be exchanged between the loops using local variables.

Figure 13 shows the block diagram of a simple example. The value of the
gain $K$ is generated in the while loop, being used in the simulation loop via a local variable.

Note how both loops in the example are stopped by only one Stop button. A local variable is used, and a boolean constant drawn to another local variable causes the while loop to halt when the Halt button in the simulation diagram is pressed. To use the local variable of a stop (halt) boolean variable, the mechanical action of the original stop button must not be set as a latch operation, but instead as a switch operation, e.g. Switch When Pressed. (The mechanical action can be set via the right-click menu on the boolean stop control.)

3.7 Importing models from Control Design Toolkit and System Identification Toolkit

3.7.1 Importing models from Control Design Toolkit

You can import/export models between the Simulation Module and the Control Design Toolkit using the conversion functions on the Control Design Toolkit / Model Conversion palette. The two conversion functions,

- Convert Control Design to Simulation, and
- Convert Simulation to Control Design

are shown in Figure 14.
3.8 Translating SIMULINK models into LabVIEW Simulation models

You can translate SIMULINK\(^4\) models into LabVIEW simulation models using the menu Tools / Simulation Tools / SIMULINK Translator.

References


\(^4\)Produced by The MathWorks.
Figure 14: The conversion functions on the Control Design Toolkit / Model Conversion palette