LabVIEW™

Simulation Module User Manual

November 2005
371013C-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222, Thailand 662 278 6777, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on `ni.com/legal` for more information about National Instruments trademarks.

MATLAB®, Stateflow®, and Simulink® are the registered trademarks of The MathWorks, Inc. Further, other product and company names mentioned herein are trademarks, registered trademarks, or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS.

BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions ix
Related Documentation x

Chapter 1
Introduction to Simulation

Dynamic System Models 1-2
Physical Models 1-2
Lumped versus Distributed Parameter Models 1-2
Linear versus Nonlinear Models 1-3
Time-Variant versus Time-Invariant Models 1-3
Continuous versus Discrete Models 1-3
Empirical Models 1-4
Linear Model Forms 1-4
Ordinary Differential Equation Solvers 1-4
Rapid Control Prototyping and Hardware-in-the-Loop Configurations 1-5

Chapter 2
Building Simulations

Using the Simulation Loop 2-1
Displaying Additional Inputs 2-2
Removing and Rearranging Inputs 2-2
Configuring Simulation Parameters 2-3
Configuring Simulation Parameters Interactively 2-3
Configuring Simulation Parameters Programmatically 2-4
Using Simulation Functions 2-4
Defining Feedthrough Behavior and Feedback Cycles 2-5
Changing Icon Styles 2-8
Configuring Simulation Functions 2-8
Configuring Discrete Simulation Functions 2-10
Stopping a Simulation Programmatically 2-11
Placing LabVIEW VIs, Functions, and Structures on the Simulation Diagram 2-11

Defining Linear Models 2-12
Defining Linear Models Interactively 2-12
Defining Linear Models Programmatically 2-14
Defining Linear Models Using a Constant 2-14
Defining Linear Models Using the LabVIEW Control Design Toolkit 2-15
Transferring Linear Model Definitions between Functions 2-15

Chapter 3
Creating Simulation Subsystems

Creating and Running Subsystems 3-1
Creating Stand-Alone Subsystems 3-1
Running Subsystems within a Simulation Diagram 3-3
Defining the Feedthrough Behavior of Subsystems 3-4
Linearizing a Subsystem 3-4
Linearizing a Subsystem Interactively 3-5
Linearizing a Subsystem Programmatically 3-5
Trimming a Subsystem 3-7

Chapter 4
Executing Real-Time Applications

Determinism 4-1
Case Study: Rapid Control Prototype and Hardware-in-the-Loop Configurations 4-2
Offline Simulation 4-3
Rapid Control Prototype Configuration 4-3
Hardware-in-the-Loop Configuration 4-4
Executing Simulations on ETS Targets 4-5
Executing Simulations on RTX Targets 4-5

Chapter 5
Solving Ordinary Differential Equations

Simulation Discontinuities 5-2
ODE Solver Order and Simulation Accuracy 5-2
Variable Step-Size ODE Solvers versus Fixed Step-Size ODE Solvers 5-3
Single-Step ODE Solvers versus Multi-Step ODE Solvers 5-4
Stiff Problems 5-4
Simulation Module ODE Solvers 5-5

Chapter 6
Optimizing Design Parameters

Constructing the Dynamic System Model 6-3
Defining a Cost Function 6-4
Defining Inequality Constraints 6-7
Defining Parameter Bounds 6-9
Defining Initial Parameter Values and a Mesh 6-10
Executing the SQP Algorithm 6-11
Case Study: Designing a PID Controller for a Second-Order System 6-12

Chapter 7
Using the Simulation Translator

Converting Models into LabVIEW Code 7-1
Common Warnings 7-2

Appendix A
Technical Support and Professional Services

Glossary

About This Manual

This manual contains information about the purpose of simulation and the simulation process. This manual also describes how to use the LabVIEW Simulation Module to simulate the behavior of a dynamic system.

Use this manual to learn how to use the Simulation Module in real-time applications and how to use the Simulation Translator to convert model (`.mdl`) files developed in The MathWorks, Inc. Simulink® simulation environment into LabVIEW VIs. This manual also describes factors to consider when you develop a model and factors to consider when you select an ordinary differential equation (ODE) solver to use for a simulation.

This manual requires that you have a basic understanding of the LabVIEW environment. If you are unfamiliar with LabVIEW, refer to the *Getting Started with LabVIEW* manual before reading this manual.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names; dialog box names; and pages, sections, and components of dialog boxes.
italic	Italic text denotes variables, emphasis, or a cross reference. Italic text also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

Related Documentation

The following documents contain information that you might find helpful as you read this manual.

- *LabVIEW Help*
- *LabVIEW Control Design Toolkit User Manual*
- *LabVIEW System Identification Toolkit User Manual*
- *LabVIEW Real-Time Module Help*
- *LabVIEW Execution Trace Toolkit User Guide*
- *NI-CAN Hardware and Software Manual*
- *NI-DAQmx Help*

The following books contain information that pertains to simulating dynamic systems.

- Ascher, Uri M., and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics, 1998.
- Dorf, Richard C., and Robert H. Bishop. *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2001.
- Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.
- Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.
- Nise, Norman S. *Control Systems Engineering*, 3rd ed. New York: John Wiley & Sons, Inc., 2000.
- Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, Inc. 2001.
- Shampine, Lawrence F. *Numerical Solution of Ordinary Differential Equations*. New York: Chapman & Hall, Inc., 1994.

1

Introduction to Simulation

Simulation is a process that involves using software to recreate and analyze the behavior of dynamic systems. You use the simulation process to lower product development costs by accelerating product development. You also use the simulation process to provide insight into the behavior of dynamic systems you cannot replicate conveniently in the laboratory. For example, simulating a jet engine saves time, labor, and money compared to building, testing, and rebuilding an actual jet engine. Figure 1-1 shows a sample dynamic system.

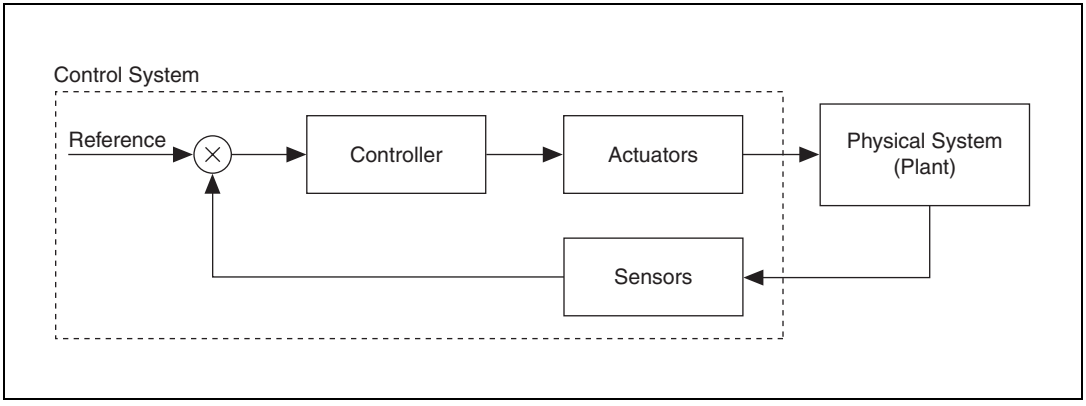

Figure 1-1. Dynamic System

The dynamic system in Figure 1-1 represents a closed-loop system, also known as a feedback system. In closed-loop systems, the controller monitors the output of the plant and adjusts the actuators to achieve a specified response. You can use the LabVIEW Simulation Module to simulate a dynamic system or a component of a dynamic system. For example, you can simulate only the plant while using hardware for the controller, actuators, and sensors.

This chapter provides an overview of the simulation process and describes how to use the Simulation Module to simulate a dynamic system.

Dynamic System Models

The simulation process relies upon the concept of a dynamic system model. A dynamic system model is a mathematical representation of the dynamics between the inputs and outputs of a dynamic system. You generally represent dynamic system models with differential equations or difference equations. You can use the LabVIEW System Identification Toolkit to obtain a model of a dynamic system.

Note This document is not intended to provide a comprehensive discussion of dynamic system models. Refer to the following books for more information about modeling: *Modern Control Systems*[1], *Feedback Control of Dynamic Systems*[2], *Digital Control of Dynamic Systems*[3], *Control Systems Engineering*[4], and *Modern Control Engineering*[5].

You can use physical laws or experimental data to develop a dynamic system model. The following sections describe features of both the physical modeling and the empirical modeling techniques.

Physical Models

The laws of physics define the physical model of a system. The following sections describe various classifications and features of physical models.

Lumped versus Distributed Parameter Models

If you can use an ordinary differential equation to describe a physical system, the resulting model is a lumped parameter model. If you can use a partial differential equation to describe a system, the resulting model is a distributed parameter model.

[1] Dorf, Richard C., and Robert H. Bishop. *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2001.

[2] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[3] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

[4] Nise, Norman S. *Control Systems Engineering*, 3rd ed. New York: John Wiley & Sons, Inc., 2000.

[5] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, Inc. 2001.

Linear versus Nonlinear Models

Dynamic system models are either linear or nonlinear. A linear model obeys the principle of superposition. The following equations are true for linear models.

$$y_1 = f(x_1)$$

$$y_2 = f(x_2)$$

$$Y = f(x_1 + x_2) = y_1 + y_2$$

Conversely, nonlinear models do not obey the principle of superposition. Nonlinear effects in real-world systems include saturation, dead-zone, friction, backlash, and quantization effects; relays; switches; and rate limiters. Many real-world systems are nonlinear, though you can linearize nonlinear models to simplify a design or analysis procedure. Refer to the *Linearizing a Subsystem* section of Chapter 3, *Creating Simulation Subsystems*, for information about linearizing a nonlinear model.

Time-Variant versus Time-Invariant Models

Dynamic system models are either time-variant or time-invariant. The parameters of a time-variant model change with time. For example, you can use a time-variant model to describe an automobile. As fuel burns, the mass of the vehicle changes with time.

Conversely, the parameters of a time-invariant model do not change with time. For an example of a time-invariant model, consider a simple robot. Generally, the dynamic characteristics of robots do not change over short periods of time.

Continuous versus Discrete Models

Dynamic system models are either continuous or discrete. Continuous models represent real-world signals that vary continuously with time. You use differential equations to describe continuous systems. For example, a model that describes the orbital motion of a satellite is a continuous model.

Conversely, discrete models represent signals that you sample at separate intervals in time. You use difference equations to describe discrete systems. For example, a digital computer that controls the altitude of the satellite uses a discrete model.

Generally, continuous system models are analog, and discrete system models are digital. Both continuous and discrete system models can be linear or nonlinear and time-invariant or time-variant.

Empirical Models

Empirical models use data gathered from experiments to define the mathematical model of a system. To some degree, physical models are empirical because you experimentally determine certain constants used to develop the model. A variety of empirical modeling methods exist. One method of empirical modeling uses tables of experimental data that represent the system you want to model. Another method for developing models uses system identification methods. System identification methods use measured data to create differential or difference equation representations that model the data. You can use System Identification Toolkit to create models using system identification methods.

Linear Model Forms

You can use the Simulation Module to represent continuous and discrete linear models in the following three forms:

- **Transfer Function**—These models use polynomial functions to define the relationship between the inputs and outputs of a dynamic system. You analyze transfer function models in the frequency domain.
- **Zero-Pole-Gain**—These models are transfer function models that you rewrite to show the gain and the locations of the zeroes and poles of the dynamic system. You analyze zero-pole-gain models in the frequency domain.
- **State-Space**—These models represent the dynamic system in terms of physical states. Continuous state-space models use first-order differential equations to describe the dynamic system, whereas discrete state-space models use first-order difference equations. You analyze state-space models in the time domain.

Ordinary Differential Equation Solvers

Because dynamic system models consist of differential equations, you must solve these differential equations to observe the behavior of the simulated system. The Simulation Module includes ordinary differential equation (ODE) solvers that solve these equations.

ODE solvers use methods to approximate the solution to a differential equation. The ODE solvers implement these methods in a variety of ways, each with various strengths and weaknesses. Defining characteristics of an ODE solver include the following qualities:

- Accuracy or order
- Stability
- Computational speed
- Use of a fixed time step size versus a variable time step size
- Use of a single step versus multiple steps

Refer to Chapter 5, *Solving Ordinary Differential Equations*, for more information about ODE solvers.

Rapid Control Prototyping and Hardware-in-the-Loop Configurations

Rapid control prototyping (RCP) and hardware-in-the-loop (HIL) configurations involve simulating different components of a dynamic system. RCP configurations simulate the controller alongside a hardware version of the plant. HIL configurations simulate the plant alongside a hardware version of the controller. Both configurations involve running the simulated component on real-time (RT) hardware.

These configurations are beneficial because you can adapt simulated models as you proceed in the design process. For example, in an RCP configuration, you can adjust the controller model as you see the effects of the simulated controller in real time. In an HIL configuration, you can adjust the plant model as the simulation progresses.

You can use the Simulation Module in conjunction with the LabVIEW Control Design Toolkit, the LabVIEW Real-Time Module, and National Instruments RT Series hardware to design and implement RCP and HIL configurations. Refer to Chapter 4, *Executing Real-Time Applications*, for more information about RCP and HIL configurations.

2

Building Simulations

You use the LabVIEW Simulation Module to build a simulation diagram, which graphically displays a simulation model in LabVIEW. You build and execute a simulation diagram using the Simulation Loop, Simulation functions, and other LabVIEW VIs and structures.

The simulation diagram uses an ordinary differential equation (ODE) solver to compute the behavior of a simulation model. Refer to Chapter 5, *Solving Ordinary Differential Equations*, for information about the ODE solvers that the Simulation Module includes.

The simulation diagram supports standard LabVIEW debugging techniques. You can use execution highlighting, breakpoints, probes, custom probes, and single-stepping on the simulation diagram.

This chapter provides information about using the Simulation Loop and Simulation functions to design, build, and configure simulations in LabVIEW.

Using the Simulation Loop

The Simulation Loop forms the boundary of the simulation diagram and contains the parameters that define how the simulation diagram executes. Figure 2-1 shows the Simulation Loop.

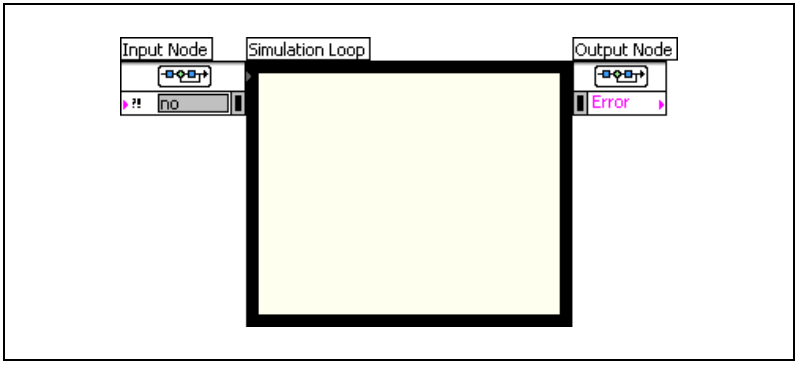

Figure 2-1. Simulation Loop

The Simulation Loop has two attached nodes. Use the Input Node of the Simulation Loop to programmatically configure parameters of a simulation. Refer to the *Configuring Simulation Parameters Programmatically* section of this chapter for information about using the Input Node to configure simulation parameters. Use the Output Node to view any errors that occur during the Simulation Loop.

The following sections provide information about adding, removing, and rearranging terminals on the Input Node.

Displaying Additional Inputs

You can use three methods to display additional inputs on the Input Node. The first method you can use is resizing the Input Node. To resize a node, first move the cursor over the Input Node to display resizing handles. Then, move the cursor over a resizing handle. Click and drag the handle down to add terminals to the node.

The second method you can use is adding individual inputs. To add the next available input, right-click the Input Node and select **Add Input** from the shortcut menu.

The third method you can use is displaying all the available inputs. To display all the available inputs, right-click the Input Node and select **Show All Inputs** from the shortcut menu.

Removing and Rearranging Inputs

You can use two methods to remove unwired inputs from the Input Node. The first method you can use is resizing the node. Click and drag the resize handle at the bottom of the node up to remove unwired inputs from the node.

The second method you can use is removing individual unwired inputs. To remove the last unwired input on the Input Node, right-click the node and select **Remove Input** from the shortcut menu.

You also can rearrange the order of unwired inputs on the Input Node. Right-click an input and select **Select Input** from the shortcut menu to view a list of inputs. The top half of this list contains the inputs currently displayed on the Input Node. The input on which you right-clicked has a checkmark by it. Select an input from the top half of the list to switch the positions of the inputs. The bottom half of this list contains the inputs not currently displayed on the Input Node. Select an input from the bottom half of the list to replace the selected input with the input you select.

Configuring Simulation Parameters

The simulation parameters define the behavior of the simulation diagram. You can configure simulation parameters using the following two methods: interactively by using the **Configure Simulation Parameters** dialog box and programmatically by wiring values to the Input Node of the Simulation Loop. You also can use a combination of these two methods in the same simulation diagram. However, values that you programmatically configure override any equivalent settings that you make in the **Configure Simulation Parameters** dialog box.

Configuring Simulation Parameters Interactively

Configure simulation parameters interactively using the **Configure Simulation Parameters** dialog box. You can launch this dialog box by double-clicking the Input Node. You also can right-click the border of the Simulation Loop and select **Configure Simulation Parameters** from the shortcut menu to launch this dialog box. When running the simulation as a stand-alone subsystem, you can select **Operate»Configure Simulation Parameters** from the pull-down menu to launch this dialog box.

The **Configure Simulation Parameters** dialog box has two pages that display different categories of parameters. You configure general simulation parameters on the **Simulation Parameters** page. You configure timing parameters on the **Timing Parameters** page. Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for information about the **Configure Simulation Parameters** dialog box.

Configuring Simulation Parameters Programmatically

Configure simulation parameters programmatically by wiring controls to the Input Node of the Simulation Loop. You also can wire the results of other VIs to the Input Node. Figure 2-2 shows how you configure a simulation diagram programmatically.

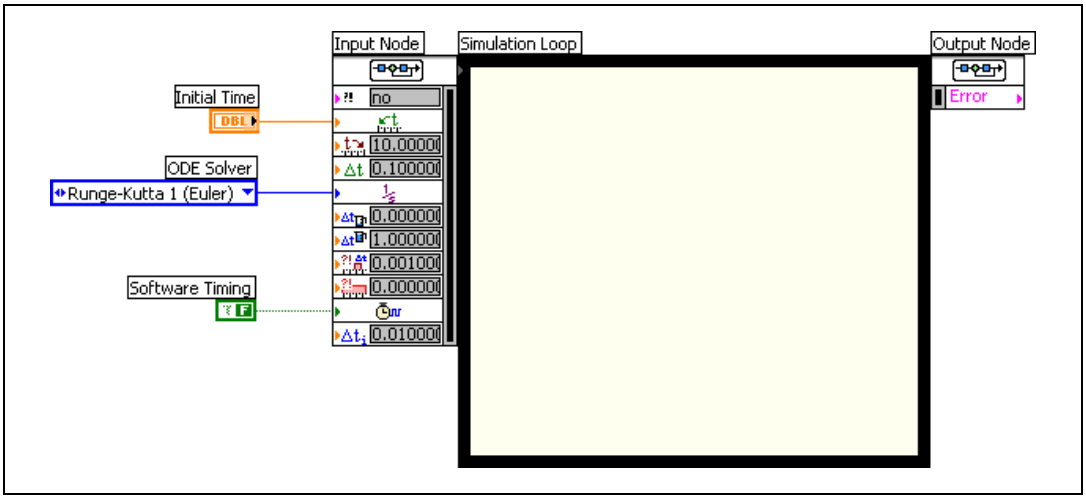

Figure 2-2. Configuring Simulation Parameters Programmatically

Figure 2-2 shows how the gray boxes on the Input Node display any values that you configure in the **Configure Simulation Parameters** dialog box. Values that you configure programmatically do not have gray boxes.

Using Simulation Functions

The Simulation functions are the elements that comprise a simulation model. You must place all Simulation functions inside a Simulation Loop or inside a simulation subsystem. Refer to Chapter 3, *Creating Simulation Subsystems*, for information about placing functions in a simulation subsystem.

The following Simulation functions are dynamic elements that depend on the ODE solver.

- Integrator
- Transfer Function
- Zero-Pole-Gain
- State-Space

Refer to Chapter 5, *Solving Ordinary Differential Equations*, for information about ODE solvers.

The following sections provide information about using Simulation functions. The following sections also describe feedback cycles, icon styles, and using LabVIEW VIs on the simulation diagram.

Defining Feedthrough Behavior and Feedback Cycles

The relationship between the inputs and outputs of a function defines the feedthrough behavior of that function. An input has direct feedthrough to an output if the function uses the input at the current step to compute the output at the current step. An input has indirect feedthrough to an output if the function does not use the input at the current step to compute the output at the current step. The indirect feedthrough function uses the input from the previous step or steps to compute the output at the current step.

This implementation of feedthrough behavior modifies the dataflow programming model LabVIEW uses. Whereas LabVIEW VIs execute after receiving values of all inputs, Simulation functions execute after receiving values of all inputs with direct feedthrough to the output. Simulation functions can execute without receiving values of inputs that have indirect feedthrough to the output.

Therefore, on a simulation diagram, you can create a feedback cycle in which data flow originates from an output of a function or subsystem that has indirect feedthrough behavior and terminates as an input of the same function or subsystem. In a feedback cycle, the output of the indirect feedthrough function or subsystem at time t is a function of the input to the same function or subsystem at time $t - dt$, $t - dt_2$, and so on.

You can use one or more Simulation functions and other LabVIEW functions in a feedback cycle as long as at least one Simulation function in the feedback cycle has indirect feedthrough behavior. The indirect feedthrough function can start the data flow by executing the function output at the current step before receiving an input from the cycle at the current step. Therefore, the input at the current step and the output at the current step must not depend on each other directly in at least one function in the cycle.

Various Simulation functions, such as the Integrator function, have indirect feedthrough behavior and therefore utilize feedback cycles. LabVIEW automatically determines the type of feedthrough behavior that exists between inputs and outputs. If you attempt to wire an output to an input that

has direct feedthrough to that output, the wire breaks. Figure 2-3 shows this behavior.

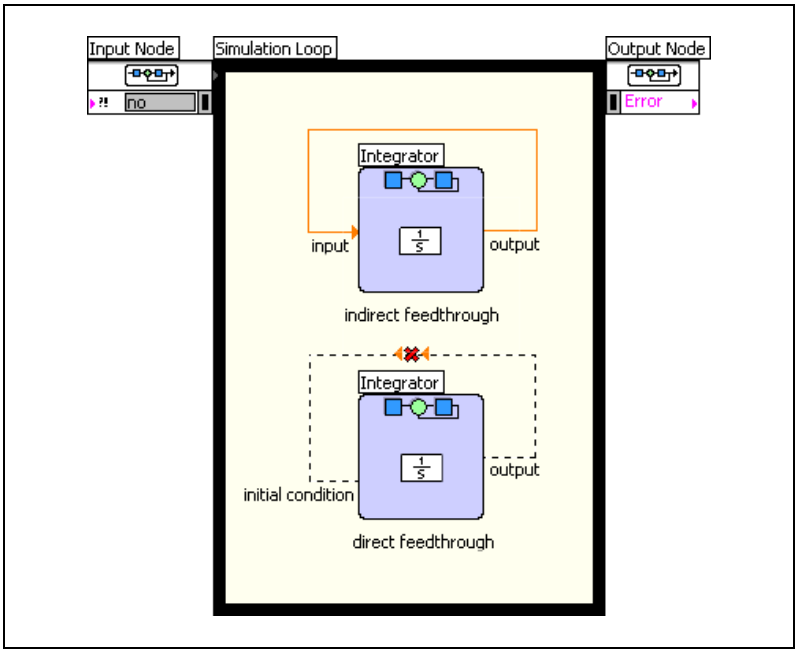

Figure 2-3. Feedback Cycles

In the first Integrator function, shown in Figure 2-3, **output** is wired to **input**. Because **input** does not have direct feedthrough to **output**, you can create a feedback cycle between **output** and **input**. In the second Integrator function, **output** is wired to the **initial condition** input. Because **initial condition** has direct feedthrough to **output**, you cannot create a feedback cycle between **output** and **initial condition**.

Note The wires on the simulation diagram use arrows to indicate the direction of data flow. These arrows help you identify feedback cycles on the simulation diagram by showing data flow direction.

Feedthrough behavior differs from function to function. The following Simulation functions have indirect feedthrough behavior:

- Integrator
- Transport Delay
- Discrete Unit Delay
- Memory

For other Simulation functions, the parameter values you specify determine the feedthrough behavior. The following Simulation functions have parameter-dependent feedthrough behavior:

- Transfer Function
- Zero-Pole-Gain
- State-Space
- Discrete Filter
- Discrete Integrator
- Discrete Transfer Function
- Discrete Zero-Pole-Gain
- Discrete State-Space

All other Simulation functions have direct feedthrough behavior. Refer to the *LabVIEW Help* for information about the direct or indirect feedthrough behavior of individual Simulation functions.

In addition to the feedback behavior of a single function, you can reverse Simulation functions to better display the data flow of an entire feedback system, as shown in Figure 2-4.

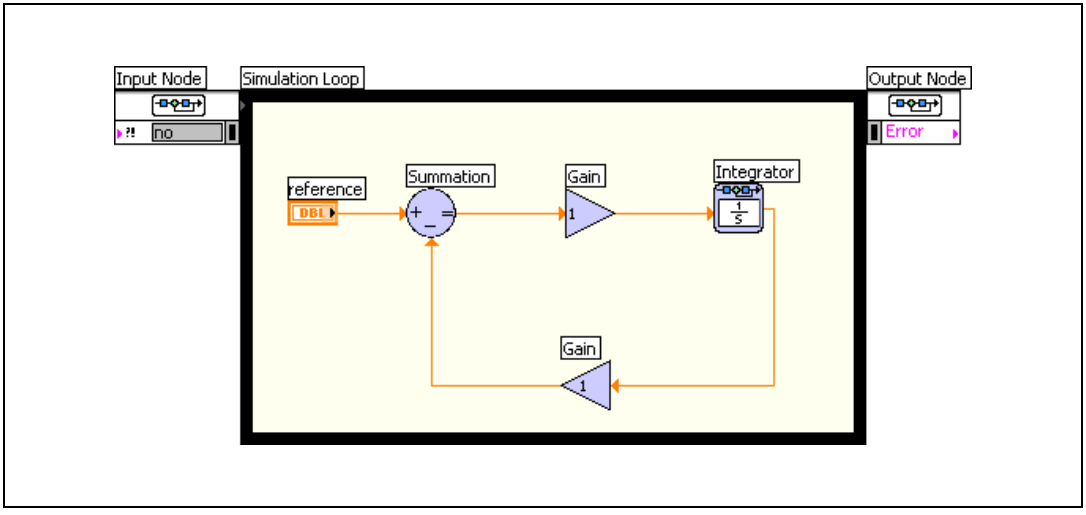

Figure 2-4. Reversing the Direction of the Gain Function

To reverse a Simulation function, right-click the icon and select **Reverse Terminals** from the shortcut menu.

Changing Icon Styles

You can change the icon style of a Simulation function on the simulation diagram. Right-click on a Simulation function and select **Icon Style** from the shortcut menu to display the following options:

- **Static**—Displays the Simulation function as a standard VI.
- **Dynamic**—Displays the Simulation function as an object that you can resize. Dynamic icons also display a preview of their contents. For example, a Sine Signal function with a dynamic icon displays a sine wave with the frequency, amplitude, and phase that you configure.
- **Text Only**—Displays the Simulation function as a list of parameter values.
- **Express**—Displays the Simulation function with a list of parameters below the icon. You can resize the parameter list to display more inputs and outputs. This icon style also shows parameter values directly on the simulation diagram.

Configuring Simulation Functions

You can configure Simulation functions using the configuration dialog box of that function. Double-click a Simulation function to launch the configuration dialog box of that function. You also can launch this dialog box by right-clicking the Simulation function and selecting **Configuration** from the shortcut menu. For example, Figure 2-5 shows the configuration dialog box for the Sine Signal function.

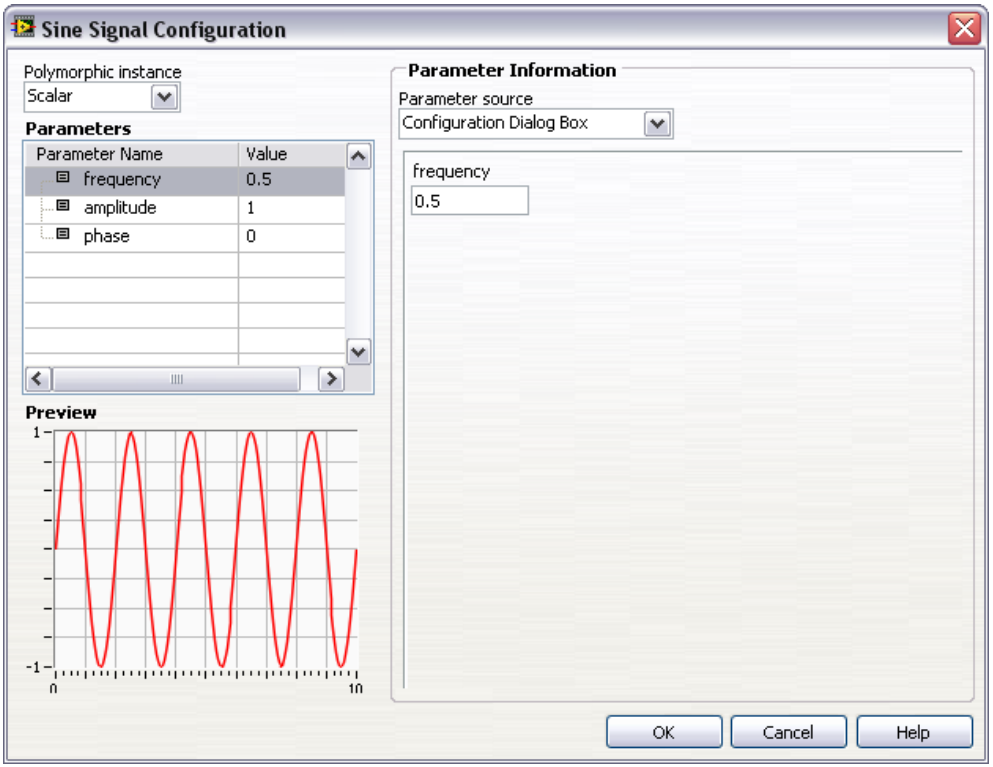

Figure 2-5. Configuration Dialog Box of the Sine Signal Function

The **Parameters** section lists all the parameters that you can configure for the Sine Signal function. When you select a parameter from the **Parameters** section, the **Parameter Information** section displays a control you can use to set the value of that parameter.

Use the **Parameter source** control to specify the source of the parameter value. If you select **Terminal**, LabVIEW displays an input terminal for that parameter on the simulation diagram, and you can wire values to this input to configure the Simulation function. If you select **Configuration Dialog Box**, LabVIEW removes that input from the simulation diagram. You then must set the value for this parameter in the configuration dialog box.

The parameters you specify for a Simulation function are unique to that function. If you create multiple instances of the same function, you can set different parameter values for each instance.

Configuring Discrete Simulation Functions

All discrete Simulation functions have a **sample period (sec)** parameter and a **sample skew (sec)** parameter. These parameters are located in the configuration dialog box of that function. The **sample period (sec)** parameter sets the length of the time step of that function. The **sample skew (sec)** parameter delays the execution of that time step. Figure 2-6 shows how these two parameters affect the execution of a discrete Simulation function.

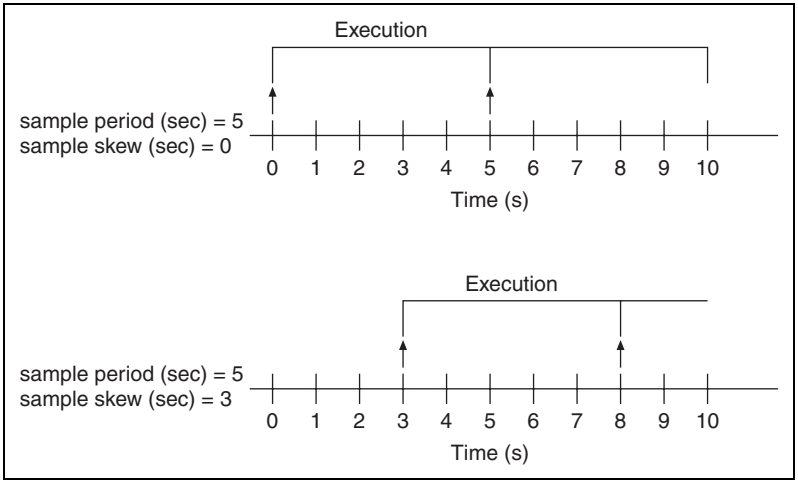

Figure 2-6. How Period and Skew Affect a Discrete Simulation Function

The **sample period (sec)** of a discrete function must be a multiple of the discrete time step of the simulation. To configure the discrete time step, double-click the Simulation Loop to launch the **Configure Simulation Parameters** dialog box. On the **Simulation Parameters** page, you can enter the **Discrete Time Step** or place a checkmark in the **Auto Discrete Time** checkbox.

Placing a checkmark in the **Auto Discrete Time** checkbox specifies that you want the Simulation Module to calculate the discrete time step of the simulation based on the **sample period (sec)** parameter and **sample skew (sec)** parameter of each discrete Simulation function on the simulation diagram. The step size of a fixed step-size ODE solver also affects the discrete time step of the simulation. The step sizes of variable step-size ODE solvers do not affect this discrete time step.

The Simulation Module calculates the floating-point greatest common divisor (GCD) of two numbers, x_1 and x_2, by finding the largest value of z such that the following equation is true:

$$\left| \frac{x_i}{z} - round\left(\frac{x_i}{z}\right) \right| \leq \varepsilon, \; i = 1, 2 \ldots n$$

In this equation, n is the total number of discrete Simulation functions on the simulation diagram. x_1 is the **sample period (sec)** parameter and the **sample skew (sec)** parameter of a single discrete function on the simulation diagram. x_2 represents these parameters for a second discrete function on the simulation diagram. z is the discrete time step of the simulation, and ε is a small number. The Simulation Module uses an extension of this equation to calculate the floating-point GCD of a set of discrete functions on the simulation diagram with sample periods x_1, x_2 ... x_n, where $x_n > 0$.

Stopping a Simulation Programmatically

Use the Halt Simulation function to stop a simulation programmatically. Place the Halt Simulation function on the simulation diagram and wire a Boolean control to the **Halt?** input. If the **Halt?** Boolean control is TRUE, the function stops the simulation after the current time step. You also can place a Halt Simulation function in a simulation subsystem to stop the execution of the parent simulation diagram. The Halt Simulation function operates like the conditional terminal on a While Loop. However, you can place more than one Halt Simulation function on the simulation diagram. With multiple Halt Simulation functions, you can stop the simulation from various points in a simulation diagram or subsystem.

Placing LabVIEW VIs, Functions, and Structures on the 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 that subVI on a simulation diagram.

By default, the Simulation Module executes VIs as continuous functions. You can change this behavior by using the **SubVI Node Setup** dialog box. To launch this dialog box, right-click on a VI and select **SubVI Node Setup** from the shortcut menu. You can configure a VI to execute at only major time steps of the ODE solver, at both major and minor time steps of the ODE solver, as a discrete function, or at initialization of the simulation

diagram. Refer to the *SubVI Node Setup Dialog Box* topic of the *LabVIEW Help* for more information about configuring the behavior of VIs on the simulation diagram. Refer to Chapter 5, *Solving Ordinary Differential Equations*, for information about ODE solvers.

Defining Linear Models

You use the Continuous Linear Systems functions and the Discrete Linear Systems functions to define continuous and discrete linear system models. The Simulation Module supports transfer function, zero-pole-gain, and state-space model forms. Refer to the *Linear Model Forms* section of Chapter 1, *Introduction to Simulation*, for information about these forms. The following sections provide information about defining models interactively and programmatically.

Defining Linear Models Interactively

Use the configuration dialog box of these Simulation functions to define a model interactively. The first step in defining the model is specifying whether the model is single-input single-output (SISO) or multiple-input multiple-output (MIMO). Select the appropriate option from the **Polymorphic instance** pull-down list. Then, select the **Transfer Function** parameter from the **Parameters** listbox. The **Parameter Information** section shows the configuration options for the model.

You define the size of MIMO models using the **Inputs** and **Outputs** text boxes in the **Model Dimensions** section. If the model is state-space, you also define the number of states using the **States** text box. The **Model Dimensions** section is dimmed if you configure a SISO model because SISO models have only one input and one output. Figure 2-7 shows a sample configuration dialog box for a MIMO transfer function model.

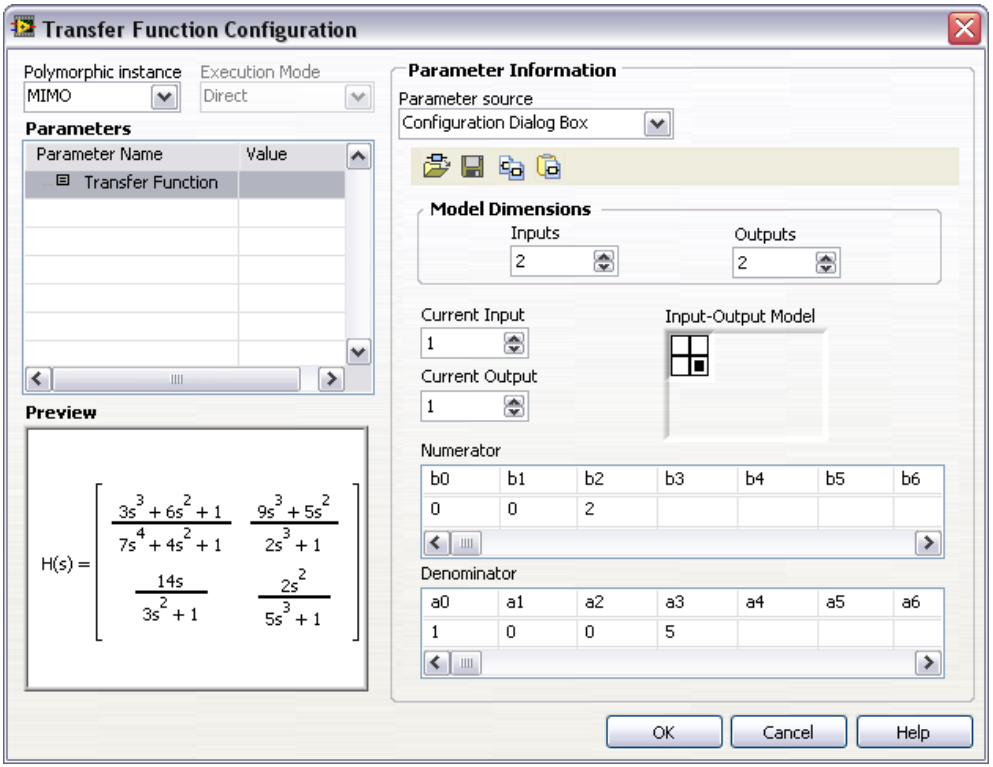

Figure 2-7. Configuring a MIMO Transfer Function Model

The example in Figure 2-7 shows a model with two inputs and two outputs. The **Inputs** and **Outputs** text boxes define these dimensions. The **Numerator** and **Denominator** vectors define the coefficients of the equation at the current input-output location. The **Current Input** and **Current Output** text boxes define the current input-output location. The **Input-Output Model** matrix also shows this location graphically with a black square.

To specify another input-output location, click an element in the **Input-Output Model** matrix or adjust the values of the **Current Input** and **Current Output** text boxes. For example, to define the bottom-left equation of the MIMO model in Figure 2-7, decrement the value of the **Current Input** text box by one. You also can click the bottom-left element of the **Input-Output Model** matrix. The **Numerator** and **Denominator** vectors then define the coefficients of the bottom-left equation.

Defining Linear Models Programmatically

You can define models programmatically using the following methods:

- Wire a constant to the input of the function.
- Wire a model you constructed with the LabVIEW Control Design Toolkit to the input of the function.

The following sections provide information about these methods.

Defining Linear Models Using a Constant

Complete the following steps to create a constant that represents the model.

1. Define the model using the steps described in the *Defining Linear Models Interactively* section of this chapter.
2. Click the **Copy** button on the toolbar of the configuration dialog box.
3. Select **Terminal** from the **Parameter source** pull-down list.
4. Click the **OK** button to close the configuration dialog box and return to the block diagram.
5. Select **Edit»Paste** from the pull-down menu to paste a constant on the block diagram. This constant contains the terms you defined in step 1.
6. Wire the output of the constant to the appropriate input of the Simulation function.

Figure 2-8 shows a constant that defines a transfer function model.

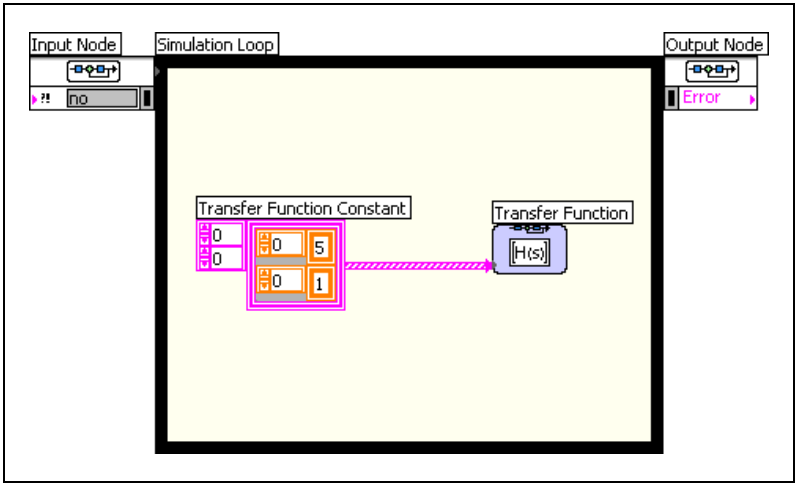

Figure 2-8. Wiring a Constant to Define a Transfer Function

Defining Linear Models Using the LabVIEW Control Design Toolkit

If you already have created a controller model using the Control Design Toolkit, you can place that model on the block diagram and wire the output of the model to the input of a Simulation function. The **Parameter source** pull-down list on the configuration dialog box of the Simulation function must be set to **Terminal**. Figure 2-9 shows a transfer function controller model wired to a Simulation function.

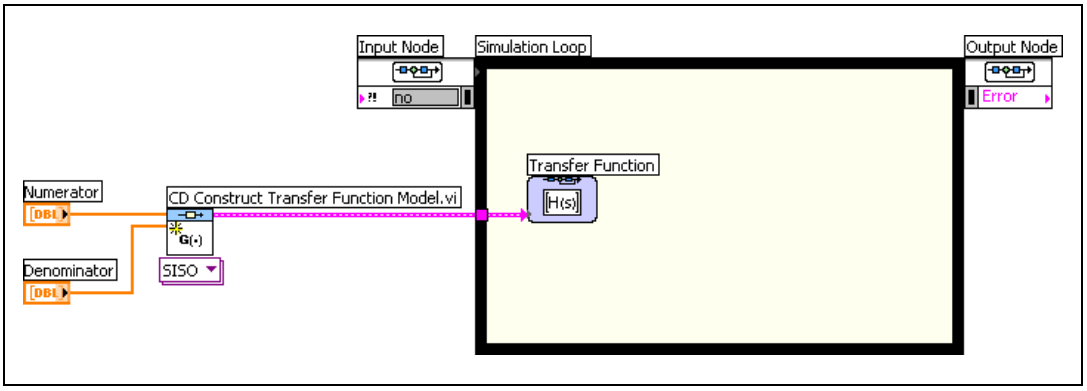

Figure 2-9. Wiring a Controller Model to Define a Transfer Function

You can use the Model Conversion VIs in the Control Design Toolkit to convert model forms and representations on the simulation diagram. For example, you can wire a continuous controller model to the input of the CD Convert Continuous to Discrete VI. Then, wire the output of this VI to the input of a discrete Simulation function.

Transferring Linear Model Definitions between Functions

If you define a model interactively as described in the *Defining Linear Models Interactively* section of this chapter, you can transfer that model definition to another Simulation function using the following two methods:

- Use the **Copy** and **Paste** buttons on the toolbar of the configuration dialog box. After defining a model, click the **Copy** button to copy that definition to the clipboard. Then, place another Simulation function on the simulation diagram, launch the configuration dialog box for that function, and click the **Paste** button.
- Use the **Save** and **Load** buttons on the toolbar of the configuration dialog box. After defining a model, click the **Save** button to save that definition to a data file. This data file is compatible with the LabVIEW

Control Design Assistant. Then, place another Simulation function on the simulation diagram, launch the configuration dialog box for that function, and click the **Load** button to specify the data file you saved. Use this method to transfer a model definition to another computer.

The following rules apply when transferring model definitions between functions.

- If you transfer a SISO model definition to a MIMO Simulation function or vice versa, the Simulation Module populates only the first input-output pair of the target Simulation function.
- You cannot transfer model definitions to another model form. For example, if you save a transfer function model definition to a data file, you cannot load that data file into a State-Space function.

3

Creating Simulation Subsystems

Simulation diagrams can require a large amount of space on the block diagram. To reduce the amount of space required to create a simulation diagram, you can convert a section of that simulation diagram into a simulation subsystem. You also can create a new VI to use as a simulation subsystem later.

Simulation subsystems are similar to LabVIEW subVIs because you can create subsystems to reuse portions of code and simplify common tasks. Refer to the *Creating SubVIs* topic of the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for information about subVIs.

This chapter provides information about creating, configuring, and running simulation subsystems. This chapter also describes how to linearize and trim a simulation subsystem.

Creating and Running Subsystems

You can run a simulation subsystem as a stand-alone VI. You also can place a simulation subsystem within the simulation diagram of another VI. The LabVIEW Simulation Module provides a method of creating a subsystem for each of these situations. Although you might create a subsystem for one situation, you can configure that subsystem to run in the other situation. For example, if you create a subsystem to run as a stand-alone VI, you later can place that subsystem within the simulation diagram of another VI.

The following sections provide information about these methods.

Creating Stand-Alone Subsystems

Complete the following steps to create a simulation subsystem that runs as a stand-alone VI.

1. Launch LabVIEW.

2. Select **File»New** from the pull-down menu.
3. Select **Other Files»Simulation Subsystem** from the **Create New** tree.
4. Click the **OK** button. LabVIEW creates a new VI with a yellow block diagram that represents a simulation diagram.
5. Create the simulation diagram code and configure simulation parameters. Because this new subsystem has no Simulation Loop, you configure simulation parameters, such as the ordinary differential equation (ODE) solver, by selecting **Operate»Configure Simulation Parameters** from the pull-down menu.

   You also can configure properties that affect how a stand-alone subsystem appears and runs. To configure these properties for a stand-alone subsystem, select **File»VI Properties** from the pull-down menu to launch the **VI Properties** dialog box. Refer to the *LabVIEW Help* for more information about the VI properties you can configure using the **VI Properties** dialog box.
6. Save the simulation subsystem. You can run this subsystem by running the VI.

Although this subsystem now is configured to run as a stand-alone VI, you can place this subsystem within the simulation diagram of another VI. If you also want to wire controls and indicators to subsystem inputs and outputs, you must build a connector pane for the subsystem manually. The connector pane defines the relationships between block diagram terminals and subsystem inputs and outputs. After you define a connector pane for a subsystem, the Simulation Module creates a configuration dialog box for that subsystem.

Refer to the *Placing SubVIs on Block Diagrams* topic of the *LabVIEW Help* for information about placing subsystems within other VIs. Refer to the *Creating SubVIs* topic of the *LabVIEW Help* for information about setting up a connector pane. Refer to the *Configuring Simulation Functions* section of Chapter 2, *Building Simulations*, for information about using the configuration dialog box.

As you create the connector pane, consider the parameter requirements and the initial value of each parameter. The default source for a subsystem parameter depends on whether you specify that parameter as required, recommended, or optional. The initial value for the parameter is the default value of the parameter control.

The following list describes the availability and default sources of parameters.

- If the connection is required, the parameter is available only as a terminal on the simulation diagram. The parameter is not visible in the configuration dialog box.
- If the connection is recommended, the default source of the parameter is the terminal. You also have the option to configure the parameter using the configuration dialog box.
- If the connection is optional, the default source of the parameter is the configuration dialog box. You also have the option to configure the parameter using a terminal on the node.

Running Subsystems within a Simulation Diagram

Complete the following steps to create a simulation subsystem that runs within another simulation diagram.

1. Launch LabVIEW and create a new blank VI.
2. Place a Simulation Loop on the block diagram.
3. Create the simulation diagram code and configure simulation parameters using the **Configure Simulation Parameters** dialog box. Refer to the *Configuring Simulation Parameters* section of Chapter 2, *Building Simulations*, for information about using this dialog box to configure simulation parameters.
4. Select a section of simulation diagram code and select **Edit»Create Simulation Subsystem** from the pull-down menu. LabVIEW replaces the code you selected with a single node that represents the simulation subsystem. The node icon shows the block diagram of the simulation subsystem.
5. Run the subsystem by running the VI that contains the subsystem.

Note When you create a subsystem using this method, the Simulation Module automatically creates a connector pane and configuration dialog box for the simulation subsystem. To launch this configuration dialog box, double-click the subsystem icon. To edit the front panel or block diagram of the subsystem, right-click the subsystem icon and select **Open Subsystem** from the shortcut menu.

Subsystems within a simulation diagram inherit the simulation parameters, such as the ODE solver to use, of that simulation diagram. These subsystems also inherit properties, such as those you set using the **VI Properties** dialog box, from the parent VI. If you want to run that

subsystem as a stand-alone VI, you must open the subsystem and select **Operate»Configure Simulation Parameters** or **File»VI Properties** from the pull-down menu.

When you are debugging a subsystem that is within another simulation diagram, you cannot use execution highlighting, breakpoints, probes, or single-stepping. You also cannot step into a subsystem. You can set a breakpoint on the entire subsystem by right-clicking the subsystem and selecting **Set Breakpoint** from the shortcut menu. You also can use a probe or a custom probe to monitor the subsystem output.

Note You cannot create a polymorphic simulation subsystem.

Defining the Feedthrough Behavior of Subsystems

LabVIEW flattens the hierarchy of a simulation subsystem when you run that subsystem. This flattened hierarchy means Simulation functions inside a simulation subsystem can execute when that function receives all the inputs that function uses, even if the subsystem did not receive all the inputs the subsystem uses. A simulation subsystem saves a feedthrough mapping from all inputs to all outputs. If a subsystem input does not depend on the subsystem output at the current time step, you can use the subsystem as an indirect feedthrough function in a feedback cycle. Refer to the *Defining Feedthrough Behavior and Feedback Cycles* section of Chapter 2, *Building Simulations*, for more information about feedback cycles.

Linearizing a Subsystem

Linearizing a continuous nonlinear subsystem involves approximating the behavior of the subsystem around an operating point. The operating point is the set of the inputs and states of the subsystem. When you linearize a subsystem, the result is a linear time-invariant (LTI) state-space model. You can design a controller for LTI models using the LabVIEW Control Design Toolkit. Refer to the *Linear versus Nonlinear Models* section of Chapter 1, *Introduction to Simulation*, for information about linear and nonlinear models.

Before you linearize a subsystem, you must choose the subsystem inputs and outputs to include in the LTI model. If an input or output is constant, you can exclude that input or output from the LTI model. You cannot exclude a subsystem state from the LTI model.

You also can change the value of subsystem inputs and states. You cannot change the value of subsystem outputs, because outputs are functions of the inputs and states of the subsystem. However, you can trim a subsystem to specify certain output conditions. Refer to the *Trimming a Subsystem* section of this chapter for information about trimming a subsystem.

You can linearize a subsystem interactively by using the **Linearize Subsystem** dialog box or programmatically by using the SIM Linearize VI. The following sections provide information about using these two methods to linearize a subsystem.

Linearizing a Subsystem Interactively

You can linearize a subsystem using the **Linearize Subsystem** dialog box. You use this dialog box to select the inputs and outputs of a subsystem that you want to include in the LTI model. To launch this dialog box, select **Tools»Control Design and Simulation»Linearize Subsystem** from the pull-down menu. Refer to the *LabVIEW Help* for more information about using the **Linearize Subsystem** dialog box.

Linearizing a Subsystem Programmatically

You can linearize a subsystem using the SIM Linearize VI. By default, this VI includes all subsystem states, inputs, and outputs in the LTI model. You can exclude inputs and outputs from the LTI model using the SIM Query Subsystem VI and the SIM Set Parameter Value VI. You also use these VIs to change the value of an input or state.

First, wire a path or reference to a subsystem to the **Path** input of the SIM Query Subsystem VI. This VI returns the **States**, **Inputs**, and **Outputs** of the subsystem. Second, wire the **States**, **Inputs**, or **Outputs** parameter to the **Parameters In** input of the SIM Set Parameter Value VI. You must use one SIM Set Parameter Value VI for each parameter on which you want to operate. Use the this VI to change the LTI model in the following ways:

- **Exclude an input or output from the LTI model**—Set the **Parameter Type** of that input or output to **Static**. Because you cannot exclude a state from the LTI model, changing the **Parameter Type** of a state to **Static** does not change the LTI model.
- **Include an input or output in the LTI model**—Set the **Parameter Type** of that input or output to **Variable**.

- **Change the value of an input or state that is included in the LTI model**—Change the **Value** of that input or state. Because subsystem inputs and states influence the output, directly changing the **Value** of an output does not change the LTI model.

Note The default **Parameter Type** of each state, input, and output is **Variable**.

Finally, wire the modified **States**, **Inputs**, or **Outputs** parameter to the appropriate input of the SIM Linearize VI. You also must wire a path or reference to the subsystem to this VI. The resulting **State-Space Model** includes all the subsystem states and any inputs and outputs you set to **Variable**. The **State-Space Model** does not include any inputs and outputs you set to **Static**.

For example, consider a subsystem that is a nonlinear model of a car, `car.vi`. If you want to approximate the behavior of this model when the engine speed of the car is 50 miles per hour, linearize the model using a value of 50 mph for the `engine speed` parameter. Also consider any parameters you want exclude from the LTI model. For example, you generally do not need to account for gravitational acceleration in an LTI model because gravitational acceleration is a constant. Figure 3-1 shows a LabVIEW block diagram that linearizes a car model using these specifications.

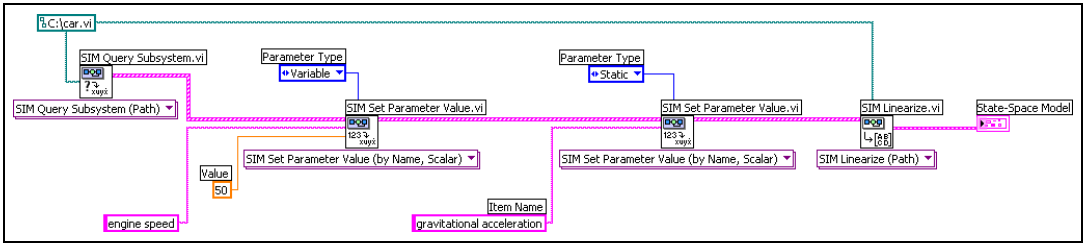

Figure 3-1. Linearizing a Nonlinear Subsystem

The example in Figure 3-1 executes the following steps:

1. Obtains the value of the **Inputs** parameter of the car model using the SIM Query Subsystem VI. The **Inputs** parameter contains subsystem inputs such as `engine speed` and `gravitational acceleration`.
2. Wires the **Inputs** parameter to the **Parameters In** input of the SIM Set Parameter Value VI. This VI sets the **Value** of `engine speed` to `50` and sets the **Parameter Type** to **Variable**.

3. Wires the modified **Inputs** parameter to the **Parameters In** input of another SIM Set Parameter Value VI. This VI sets the **Parameter Type** of `gravitational acceleration` to **Static**. Because this example does not specify a **Value** for `gravitational acceleration`, the SIM Set Parameter Value VI does not change the value of `gravitational acceleration`. Therefore, the LTI model uses the default value of `gravitational acceleration` that the SIM Query Subsystem VI returned.
4. Wires the modified **Inputs** parameter to the **Inputs** input of the SIM Linearize VI. This VI returns an LTI model, **State-Space Model**, that approximates the behavior of the car model when the engine speed is 50 mph.

Note You can specify the subsystem to linearize using a **Path** or **Reference** to the subsystem. Using a path to a subsystem causes the subsystem to load into memory every time a VI accesses the subsystem. Using a reference to a subsystem ensures that the subsystem loads into memory only once.

To obtain the parameter names or parameter values for a subsystem, use the SIM Get Parameter Names VI or the SIM Get Parameter Value VI, respectively. Refer to the *LabVIEW Help* for more information about the Trim & Linearize VIs.

Trimming a Subsystem

Use the SIM Trim VI to trim a continuous simulation subsystem. Trimming a subsystem involves searching for values of subsystem inputs and states that satisfy any conditions you specify. By default, this VI trims a subsystem to a steady state in which all state derivatives are zero. By default, this VI does not specify any conditions and includes all inputs and states in the search. You can use the SIM Query Subsystem VI and the SIM Set Parameter Value VI to change the behavior of SIM Trim VI in the following ways:

- **Trim a subsystem to a transient state**—Specify a non-zero **Value** for one or more state derivatives. You also must set the **Parameter Type** of these state derivatives to **Fixed**.
- **Specify one or more state, input, and/or output conditions**—Specify a **Value** for one or more states, inputs, and/or outputs. You also must set the **Parameter Type** of these states, inputs, and/or outputs to **Fixed**.

- **Change the initial search location**—Specify a **Value** for one or more states, inputs, and/or outputs. The values of these parameters form the location at which the SIM Trim VI begins the search. You also must set the **Parameter Type** of these states, inputs, and/or outputs to **Variable**.
- **Exclude a state and/or input from the search**—Specify a **Value** for each input and/or state that you want to exclude. You also must set the **Parameter Type** of these states and/or inputs to **Static**.

Refer to the *Linearizing a Subsystem Programmatically* section of this chapter for information about using the SIM Query Subsystem VI and the SIM Set Parameter Value VI to change the **Value** and/or **Parameter Type** of subsystem states, inputs, outputs, and state derivatives. Refer to the *SIM Set Parameter Value* topic of the *LabVIEW Help* for information about the difference between parameter types.

For example, consider a car model `car.vi` that contains `car position` as a state. The car model has a cruise-control system that must set the position of the accelerator to maintain forward movement at a specified velocity. You can specify the velocity of the car using the derivative of `car position`. Figure 3-2 shows a LabVIEW block diagram that trims the car model using a specified velocity of 60 miles per hour.

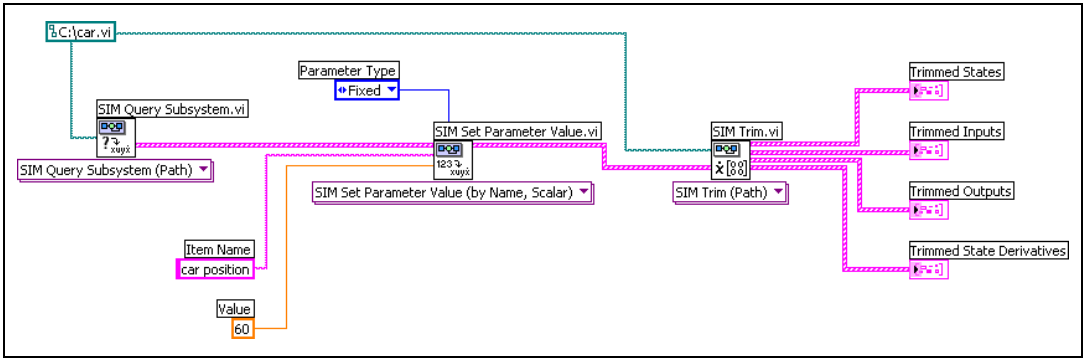

Figure 3-2. Trimming a Subsystem

The example in Figure 3-2 executes the following steps:

1. Obtains the value of the **State Derivatives** parameter of the car model using the SIM Query Subsystem VI. The **State Derivatives** parameter contains `car position`, which is equivalent to the velocity of the car.
2. Wires the **State Derivatives** parameter to the **Parameters In** input of the SIM Set Parameter Value VI. This VI sets the **Value** of `car position` to `60`. The **Parameter Type** of `car position` is **Fixed**,

which indicates this parameter is a condition that the SIM Trim VI must satisfy. This VI must return values of states and inputs that keep the velocity of the car at 60 mph.

3. Wires the modified **State Derivatives** parameter to the **State Derivatives** input of the SIM Trim VI. This VI returns the **Trimmed Inputs** and **Trimmed States** parameters that the cruise-control system must use to maintain a velocity of 60 mph. This VI also returns the **Trimmed Outputs** and **Trimmed State Derivatives** parameters that the car model returns when the cruise-control system uses the values of the **Trimmed States** and **Trimmed Inputs** parameters.

Note If the SIM Trim VI cannot satisfy all specified conditions, this VI returns the closest values to the specified conditions.

Refer to the *Linearizing a Subsystem Programmatically* section of this chapter for information about obtaining parameter names and specifying a subsystem by path or by reference. Refer to the *LabVIEW Help* for more information about the Trim & Linearize VIs.

4

Executing Real-Time Applications

You can use the LabVIEW Simulation Module with the LabVIEW Real-Time Module and various real-time (RT) targets to implement simulations and controllers in real time with real-world inputs and outputs. For example, you can combine this software and hardware to design and implement a rapid control prototype (RCP) or hardware-in-the-loop (HIL) configuration.

The Simulation Module supports RT targets running the real-time operating system of the Ardence Phar Lap Embedded Tool Suite (ETS) and RT targets using the Ardence Real-Time Extension (RTX). Refer to the *Executing Simulations on ETS Targets* section and the *Executing Simulations on RTX Targets* section of this chapter for information about executing simulations on ETS and RTX targets, respectively.

This chapter provides an overview of a real-time application and describes a case study that involves RCP and HIL configurations.

Determinism

Running a simulation or controller in real time means that the simulation time must equal the wall-clock time at each point at which the simulation or controller interacts with the real world. Generally, these physical interaction points correspond to the sampling points of the input and output hardware. Thus, at each sampling time, the simulation time must equal the wall-clock time.

To meet the real-time deadline, the software implementing the simulation or controller must execute deterministically, that is, the software must maintain a strict upper bound on the execution time of the software. Executing a model in real time requires that you use deterministic algorithms in the time-critical portion of the application. Deterministic algorithms ensure that block diagram code running at each time step meets the deadlines imposed by the timing of the hardware inputs and outputs.

To meet determinism requirements, you must use a deterministic ordinary differential equation (ODE) solver. Deterministic ODE solvers have fixed step sizes. The Simulation Module includes the following deterministic ODE solvers:

- Runge-Kutta 1
- Runge Kutta 2
- Runge-Kutta 3
- Runge-Kutta 4
- Discrete States Only

Refer to the *Simulation Module ODE Solvers* section of Chapter 5, *Solving Ordinary Differential Equations*, for more information about these and other ODE solvers.

All of the discrete ODE solvers have an inherently fixed time step size and are inherently deterministic. Therefore, the discrete ODE solvers are appropriate for real-time implementation. The Discrete Systems functions use the discrete ODE solvers in their implementation. ODE solver determinism is important only when you use continuous dynamic functions, such as the Integrator, State-Space, Transfer Function, and Zero-Pole-Gain functions.

Case Study: Rapid Control Prototype and Hardware-in-the-Loop Configurations

The following sections provide an overview of the process you might use to simulate a dynamic system. The following sections also describe an example offline simulation, RCP configuration, and HIL configuration.

Offline Simulation

The starting point is an offline simulation of the full dynamic system. Offline systems are not connected to any hardware. The simulation diagram in Figure 4-1 represents a simple control system. The system contains a controller, a model of the plant, and a front panel control that represents the set point or reference signal.

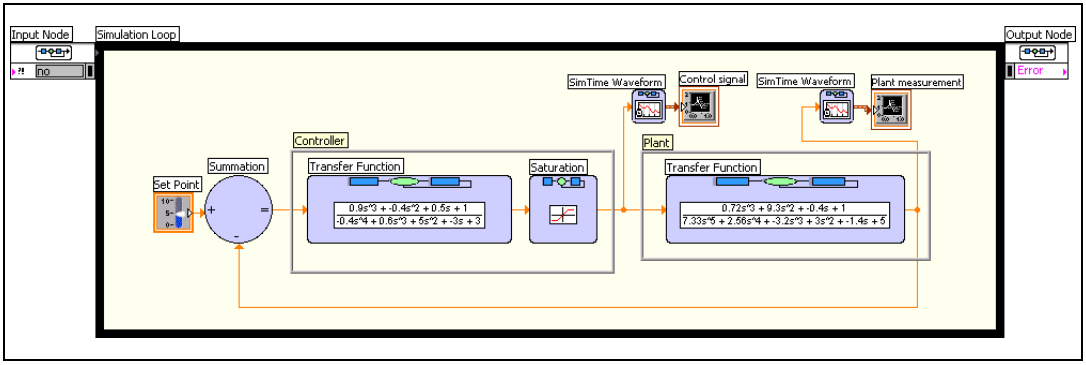

Figure 4-1. Full System Simulation

If you are running an offline simulation on a Windows operating system, National Instruments recommends you place a checkmark in the **Software Timing** checkbox on the **Timing Parameters** page of the **Configure Simulation Parameters** dialog box for optimal performance. Refer to the *Configure Simulation Parameters Dialog Box* topic in the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about the timing parameters of the Simulation Loop.

Rapid Control Prototype Configuration

An RCP configuration simulates the controller model when the controller model is connected to hardware actuators and hardware sensors. To convert an offline simulation to an RCP configuration, remove the plant model from the simulation. Replace the plant input with an output from a hardware device, and replace the plant output with an input from a hardware device.

Figure 4-2 shows an RCP configuration of the example in Figure 4-1.

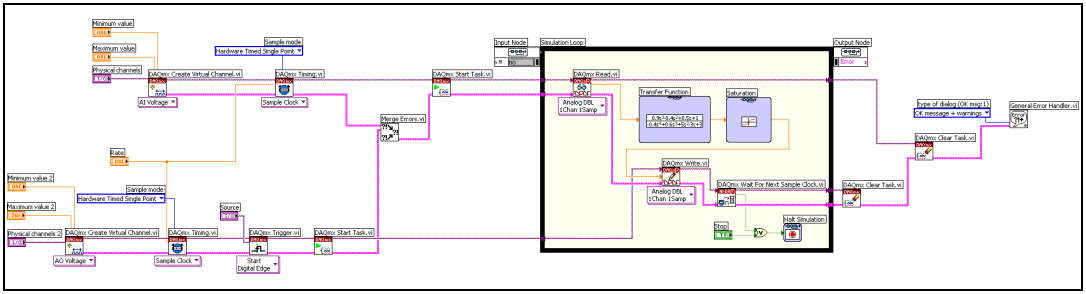

Figure 4-2. Rapid Control Prototype Configuration

The example in Figure 4-2 uses NI-DAQmx driver software to communicate with the hardware plant. The DAQmx Read VI receives an analog value from the hardware plant, and the DAQmx Write VI returns an analog value to the hardware plant.

Hardware-in-the-Loop Configuration

A HIL configuration simulates the plant model when the plant model is connected to a hardware controller. To convert an offline simulation to a HIL configuration, remove the controller model from the simulation. Replace the controller input with an output from a hardware device, and replace the controller output with an input from a hardware device. The result is a system similar to the RCP implementation, except with the controller model, not the plant model, replaced with physical hardware inputs and outputs.

Figure 4-3 shows a HIL configuration of the example in Figure 4-1.

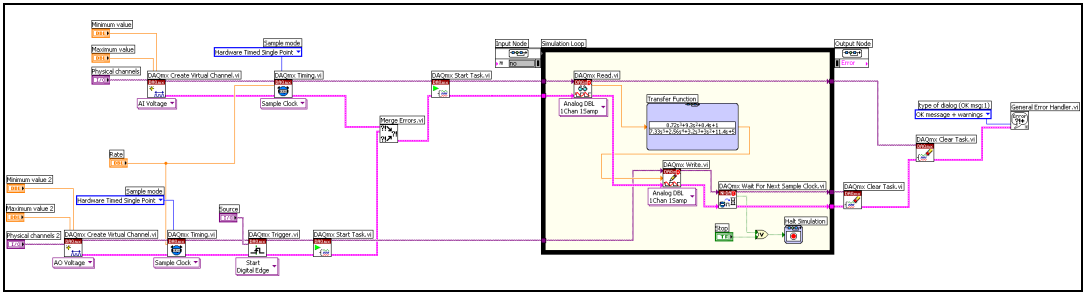

Figure 4-3. Hardware-in-the-Loop Implementation

The example in Figure 4-3 uses NI-DAQmx driver software to communicate with the hardware controller. The DAQmx Read VI receives an analog value from the hardware controller, and the DAQmx Write VI returns an analog value to the hardware controller.

Executing Simulations on ETS Targets

If you are executing a simulation on an ETS target, National Instruments recommends you let the Simulation Module calculate the necessary value of the simulation period. To automatically calculate the period, place a checkmark in the **Auto Period** checkbox, which is located on the **Timing Parameters** page of the **Configure Simulation Parameters** dialog box.

Executing Simulations on RTX Targets

If you are executing a simulation with a non-zero period on an RTX target, you must use software timing to avoid run-time errors. Place a checkmark in the **Software Timing** checkbox, which is located on the **Timing Parameters** page of the **Configure Simulation Parameters** dialog box. Then, place a subVI that contains a Wait Until Next ms Multiple function on the simulation diagram or subsystem, and set the **millisecond multiple** accordingly.

When you follow this procedure, other tasks can to continue to execute when the simulation is not scheduled to execute. Refer to the *LabVIEW Help* for more information about using the LabVIEW Real-Time Module for RTX Targets.

5

Solving Ordinary Differential Equations

To compute the behavior of a continuous model over time, the LabVIEW Simulation Module must solve the following initial value problem:

$$\frac{dy}{dt} = f(t, y)$$

$$y(t_0) = y_0$$

In these equations, y represents the outputs of the Integrator or Discrete Integrator functions, y_0 represents the initial conditions for those Integrator functions, and $f(t, y)$ represents the non-integrating functions on the simulation diagram.

The Simulation Module provides a number of solution methods for this problem. Each ordinary differential equation (ODE) solver approximates the behavior of the model at time $t + dt$ based on the behavior of the model from t_0 to t. The quantity dt is the step size of the ODE solver, and the interval from t to $t + dt$ is one time step taken by the ODE solver.

A time step at time $t + dt$ is a major time step of the ODE solver. All ODE solvers might need to evaluate the simulation diagram multiple times between major time steps to compute accurate values for time $t + dt$. A time step at these intermediate evaluation times is a minor time step of the ODE solver. Because LabVIEW indicators update only at major time steps, you might notice data flowing through the simulation diagram several times before LabVIEW updates the indicators and graphs.

Understanding the various strengths and weaknesses of the ODE solvers when simulating different types of models is useful so you can determine the appropriate ODE solver to use for an application. Refer to the following books for more information about ODE solvers: *Computer Methods for*

Ordinary Differential Equations and Differential-Algebraic Equations[1] and *Numerical Solution of Ordinary Differential Equations*[2].

Simulation Discontinuities

In general, the Simulation Module ODE solvers assume that all simulation diagram signals and signal derivatives are continuous throughout any time step. To get the most accurate solution possible, the ODE solver must stop and restart whenever the solver encounters a discontinuity. Therefore, the presence of many discontinuities in a simulation limits the maximum step size that an ODE solver can take. The number of discontinuities influences which ODE solver you choose.

LabVIEW already accounts for discontinuities that the Nonlinear Systems and Discrete Linear Systems functions introduce. However, if a model contains continuous subVIs whose outputs or output derivatives are not continuous, the model must reset the ODE solver at the points of discontinuity. To ensure the model resets the ODE solver correctly, you can use the Detect Zero Crossing function or configure the Integrator function to reset when the appropriate signal crosses zero. Refer to the *Detect Zero Crossing* and *Integrator* topics of the LabVIEW Help for more information about these configuring and using functions.

ODE Solver Order and Simulation Accuracy

To measure the accuracy of a simulation, you can measure the error introduced into the solution per time step, which is known as the local error. You also can measure the maximum difference between the computed solution and the exact solution, which is known as the global error. The amount the error changes when you vary the step size depends on the order of the ODE solver you use. If you reduce the step size of an ODE solver by a factor of λ, then the local error is reduced by approximately λ^{n+1}, where n is the order of the ODE solver. Depending on the simulation, the global error might be reduced by approximately λ^n. Reducing the error by any amount corresponds to improving the accuracy by the same amount.

For example, consider a simulation that uses a first-order ODE solver. If you run the simulation once with a step size of 0.1 and then run the simulation with a step size of 0.05, you might reduce the global error by

[1] Ascher, Uri M., and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics, 1998.

[2] Shampine, Lawrence F. Numerical Solution of Ordinary Differential Equations. New York: Chapman & Hall, Inc., 1994.

approximately 2 times ($\lambda = 2$, $n = 1$). This reduction in error doubles the accuracy of the simulation. If you use a second-order ODE solver under the same conditions, the second run might reduce the global error by approximately 4 times ($\lambda = 2$, $n = 2$). In this case, the reduction in error quadruples the accuracy of the simulation.

As these examples show, high-order ODE solvers usually are more accurate than low-order ODE solvers. In general, you can use fewer time steps and larger step sizes with a high-order ODE solver to get the accuracy you need. Using fewer time steps decreases the effects of round-off in the solution and potentially reduces the amount of time needed to compute the solution.

However, simulations using high-order ODE solvers typically require more computational resources, such as processing power and memory, per time step than simulations using low-order ODE solvers. If you are running a simulation on limited computational resources, consider using a low-order ODE solver and smaller step sizes.

Variable Step-Size ODE Solvers versus Fixed Step-Size ODE Solvers

Some of the ODE solvers the Simulation Module provides estimate the error introduced by the ODE solver at each time step. These ODE solvers then adjust the step size throughout the simulation to ensure that this per-step error remains at a given relative and absolute tolerance. For each integrator variable *y,* the ODE solver varies the step size to control the error according to the following approximation:

$$\text{per-step error} \approx |y| * \text{relative tolerance} + \text{absolute tolerance}$$

Variable step-size ODE solvers can take small time steps when the simulation variables vary rapidly and can take larger time steps when the simulation variables vary slowly. This ability to change step sizes can increase computational efficiency.

Variable step-size ODE solvers are not appropriate for deterministic real-time applications because the computational overhead of taking a time step varies over the course of a simulation. Therefore, the Simulation Module also provides fixed step-size, deterministic ODE solvers for use in real-time applications. These ODE solvers do not estimate the local per-step error and maintain a fixed step size throughout a simulation. Refer to Chapter 4, *Executing Real-Time Applications*, for information about using the Simulation Module in real-time applications.

Single-Step ODE Solvers versus Multi-Step ODE Solvers

Single-step ODE solvers approximate the behavior of the model at time $t + dt$ by taking into account only the behavior of the model between t and $t + dt$. Conversely, multi-step ODE solvers approximate the behavior at the end of the time step by taking into account the model behavior at a number of previous time steps.

To achieve high order accuracy, single-step ODE solvers might need to evaluate the simulation diagram more often per time step than multi-step ODE solvers. Therefore, single-step ODE solvers might incur a higher computational cost per step than multi-step ODE solvers. Because of this cost, multi-step ODE solvers might be able to compute an accurate solution more efficiently than single-step ODE solvers.

However, multi-step ODE solvers require a certain amount of computation to initialize. This computation takes place each time the simulation resets an Integrator function or encounters a discontinuity. A single-step ODE solver might be more efficient for simulations that reset the ODE solver often.

Stiff Problems

Certain problems, such as problems with transients that vary more quickly than the problem solution, can be difficult to solve numerically. These problems are stiff problems. When you solve stiff problems without a stiff ODE solver, you might notice that variable step-size ODE solvers take smaller and smaller time steps until the ODE solver can no longer make progress on the simulation. You also might notice an inaccurate and rapidly growing oscillatory solution no matter how small a step size you use. In this situation, you can use a stiff ODE solver to get a more accurate solution.

Simulation Module ODE Solvers

You specify the ODE solver you want by using the **Configure Simulation Parameters** dialog box or the Input Node of the Simulation Loop. Refer to Chapter 2, *Building Simulations*, for more information about the **Configure Simulation Parameters** dialog box and the Simulation Loop.

The Simulation Module includes the following ODE solvers:

- **Runge-Kutta 1 (Euler)**—A fixed step-size, single-step explicit Runge-Kutta ODE solver of first order.
- **Runge-Kutta 2**—A fixed step-size, single-step explicit Runge-Kutta ODE solver of second order.
- **Runge-Kutta 3**—A fixed step-size, single-step explicit Runge-Kutta ODE solver of third order.
- **Runge-Kutta 4**—A fixed step-size, single-step explicit Runge-Kutta ODE solver of fourth order.
- **Runge-Kutta 23**—A variable step-size, single-step explicit Runge-Kutta ODE solver of third order.
- **Runge-Kutta 45**—A variable step-size, single-step explicit Runge-Kutta ODE solver of fifth order, which uses the Dormand-Prince coefficients.
- **BDF**—A variable step-size, variable order (orders 1 through 5) implementation of the multi-step backwards difference formula (BDF), also known as Gear's Method. This method is adequate for moderately stiff problems.
- **Adams-Moulton**—A variable step-size, multi-step variable order (orders 1 through 12) implementation of the Adams-Moulton predictor-corrector pair in predict-evaluate-correct-evaluate (PECE) mode.
- **Rosenbrock**—A variable step-size, single-step explicit solver. This method is adequate for some stiff problems.
- **Discrete States Only**—A fixed step-size solver. Use this ODE solver for simulations that do not contain any continuous functions.

6

Optimizing Design Parameters

One important application of simulating dynamic system models is using the simulation to determine parameter values that maximize some measure of performance. The LabVIEW Simulation Module includes the SIM Optimal Design VI, which you can use to obtain parameters that minimize a cost function while satisfying constraints on a dynamic system. You can use this VI with both linear and nonlinear systems, although the Simulation Module includes pre-defined options for only linear systems.

Design problems can range from designing physical elements, such as springs, to designing more abstract elements such as controllers or digital filters. Correspondingly, performance specifications might range from simple mechanical limits on outputs to more sophisticated requirements such as frequency domain norms for controlled systems.

For example, when designing a suspension system for a car, you must select a stiffness constant for a spring and a damping constant for a dissipative element. The goal is to find a parameter set that provides maximum comfort. This optimal parameter set corresponds to a performance measure, such as the average deviation of the passenger from a desired height as the car travels down the road. You use parameter design to determine this optimal parameter set while taking into account the dynamics of the system and the expected operating conditions and disturbances.

You can use several techniques to determine this parameter set. For some problems, you might be able to compute the optimum analytically. However, analytical solutions typically are difficult or impossible to compute. In such cases, you can use numerical optimization instead. A powerful and general purpose numerical optimization algorithm is sequential quadratic programming (SQP). The SIM Optimal Design VI uses this algorithm. This VI provides domain-specific functions you can use to perform parameter optimization for design purposes. Specifically, you can use this VI to determine optimal parameters from finite-horizon time-domain dynamics simulations.

The following expressions define the nonlinear optimization problem.

$$min(J(p))$$
$$h_l \le H(p) \le h_u$$
$$p_l \le p \le p_u$$

where p is a parameter value, $J(p)$ is a cost function, and $H(p)$ is a set of constraints. The objective of the SQP algorithm is to minimize $J(p)$ and satisfy $h_l \le H(p) \le h_u$ while keeping p within specified minimum and maximum values.

Designing a system using the SQP algorithm involves the following steps:

1. Constructing the dynamic system model and specifying the component of that model for which you want to find optimal parameter values.
2. Defining a performance measure, also known as a cost function, you want to minimize.
3. Defining any constraints on the dynamic system that any feasible parameter values must satisfy.
4. Defining minimum and maximum values for each parameter.
5. Defining a set of initial parameter values and an initial parameters mesh, which generates additional sets of initial parameter values.
6. Executing the SQP algorithm, using the information you specified in steps 1 through 5, by running the SIM Optimal Design VI.

Note The cost function, inequality constraints, and component to optimize make up the **Problem Specification** parameter of the SIM Optimal Design VI. For each option, you can choose from pre-defined types or specify a customized version.

The following sections provide information about each of these steps, including the pre-defined and custom types of information you can specify for each step.

Constructing the Dynamic System Model

By default, the SIM Optimal Design VI computes optimal design parameters for a proportional integral derivative (PID) controller placed in a closed-loop dynamic system. Figure 6-1 shows this controller and the dynamic system structure.

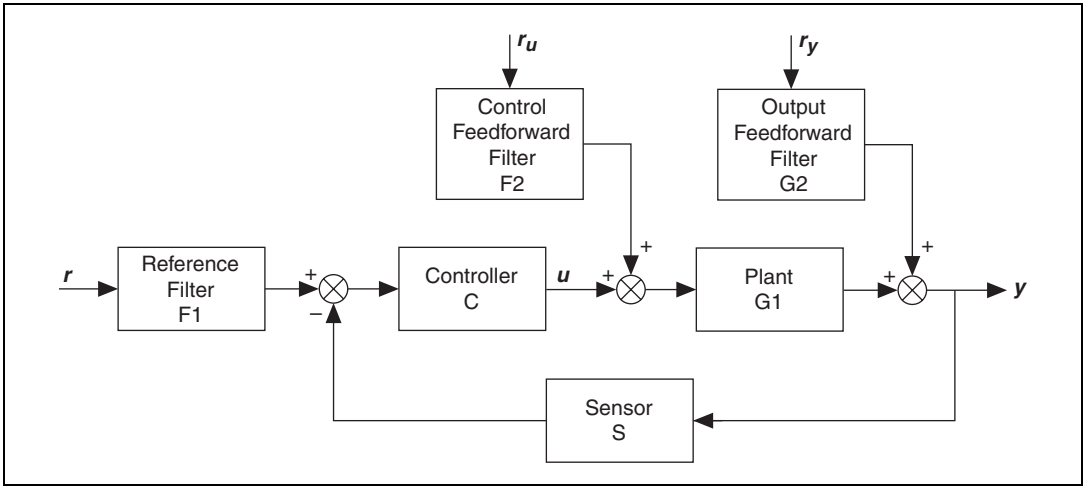

Figure 6-1. Default Dynamic System

F1, F2, C, G1, G2, and S consist of transfer functions and associated information, such as delays and sampling time. You can use the SIM Construct Default System VI to construct these transfer functions and specify reference input signals ***r***, r_u, and r_y. This VI returns the necessary dynamic system information in the **System Data** output, which you then can wire to the **System Data** input of the SIM Optimal Design VI. The SIM Optimal Design VI then excites the system using the defined inputs and obtains the time response.

Use the **System response type** parameter of the SIM Optimal Design VI to specify if you want this VI to return optimal parameter values for C, F1, or F2. By default, C is a parallel PID controller defined by the following equation:

$$U(s) = K_P e + \frac{K_I}{s} + \frac{K_D s}{\alpha s + 1}$$

where $\alpha = 0.01$, U is the control action, s is the Laplace variable, and K_P, K_I, and K_D are the proportional, integral, and derivative gains, respectively.

You also can define a custom type of system response data you want to optimize by using VI templates. To access these templates, select **File»New** from the pull-down menu to launch the **New** dialog box. Then select **VI»From Template»Simulation»Optimization Based Design** from the **Create New** tree. Double-click **SIM System response (Modify Controller Only).vit** to modify only the structure of the controller. Double-click **SIM System response (General).vit** to define a new dynamic system structure.

If you define a new dynamic system structure, the block diagram code you write must generate the output vector ***y*** and the time vector **Time**. The code also must generate the control action vector ***u*** unless the optimization problem does not require a control action. For example, if you use the SIM Optimal Design VI to design the physical parameters of a mechanism, you do not need to specify a control action. In this situation, ensure the cost function and inequality constraints you specify do not take a control action into account.

Defining a Cost Function

A cost function is the performance measure you want to minimize. Examples of cost include total power consumption, integrated error, and deviation from a reference value of a signal. The cost function is a functional equation, which maps a set of points in a time series to a single scalar value. This scalar value is the cost.

Use the **Cost type** parameter of the SIM Optimal Design VI to specify the type of cost function you want this VI to minimize. The Simulation Module includes the following types of cost functions:

- IE—A cost function that integrates the error.
- IAE—A cost function that integrates the absolute value of the error.
- ISE—A cost function that integrates the square of the error.
- ITAE—A cost function that integrates the time multiplied by the absolute value of the error.
- ITE—A cost function that integrates the time multiplied by the error.
- ITSE—A cost function that integrates the time multiplied by the square of the error.

- ISTE—A cost function that integrates the square of the time multiplied by the square of the error.
- LQ—A linear quadratic cost function.
- Sum of Variances—A cost function based on the variance of the error multiplied by the variance of the control action.

Refer to the *SIM Optimal Design* topic of the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help** from the pull-down menu, for the equations of these cost functions.

You also can define a custom cost function using a VI template. To load this template, in the **New** dialog box, select **VI»From Template»Simulation» Optimization Based Design»SIM Compute Cost.vit** from the **Create New** tree.

The block diagram of this template contains several parameters including the control action ***u***, the dynamic system output ***y***, an array of input signals, and a time series vector. You also can specify any weights on any part of the cost function.

After you define these parameters, you can write LabVIEW block diagram code to manipulate the parameters according to the cost function. For example, the following equation defines the IE cost function.

$$J_{IE} = \sum_{i} \int_{0}^{T} e_i(t)dt \cong \sum_{i} \sum_{n=0}^{N} (\Delta t \cdot e_i(n))$$

where $e(t)$ is the measured error, N is the total number of samples in the time response, n is the current time response sample, and i is the index of the current output.

Figure 6-2 shows the LabVIEW block diagram that represents this cost function.

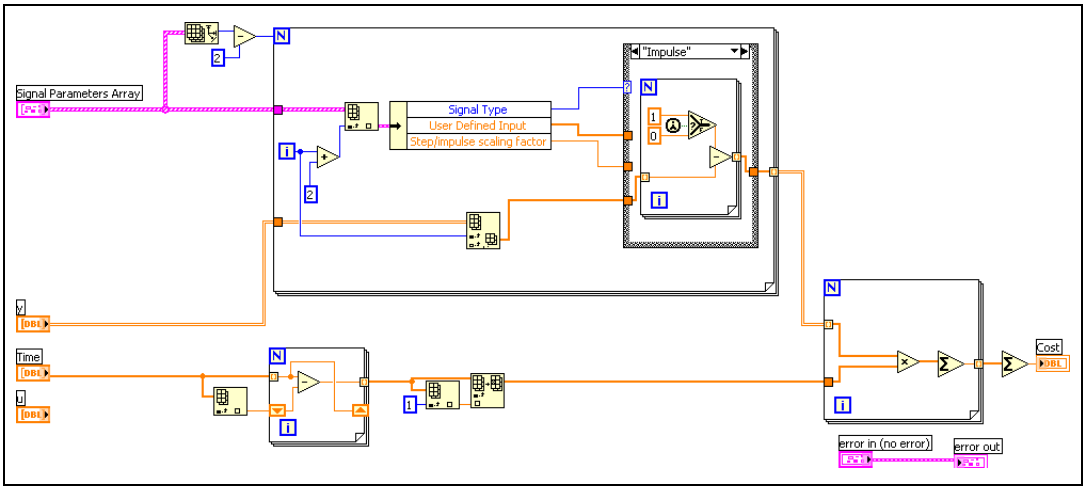

Figure 6-2. LabVIEW Block Diagram of the IE Cost Function

You can view the LabVIEW block diagrams of other cost functions in the `labview\vi.lib\addons\simulation\Optimization Based Design\Cost\` directory.

Note The IE cost function in Figure 6-2 does not take the control action into account. In this case, do not delete the **u** parameter from the block diagram. Deleting this parameter breaks the connector pane structure on which the SIM Optimal Design VI depends. Instead, leave the parameter unwired.

After you save the custom cost function as a VI, you must specify the location of the custom function in the **Problem Specification** parameter of the SIM Optimal Design VI. Select **User defined** for the **Cost type** parameter and specify the path to the VI in the **File path user defined custom cost calculation** path control.

Defining Inequality Constraints

Inequality constraints represent trade-offs implicit in the problem specification. For example, you might be able to remove error in a control loop by applying a very large control action. However, the necessary control action might be impossible to achieve in the real world. If you specify constraints on the control action before executing the SQP algorithm, you can eliminate optimal values that require an unfeasible control action.

Note Constraints add a great deal of complexity to the optimization problem. If possible, minimize the number of constraints before executing the SQP algorithm. One strategy to minimize the number of constraints involves first finding optimal values with no constraints, then gradually adding constraints and determining the least amount of constraints required for the dynamic system.

Because the optimization problem is based on a finite-horizon time-domain simulation, you specify the inequality constraints as envelopes that bound the time response of the control action and the output. You also can place inequality constraint envelopes on the rate of change of the control action and the rate of change of the output.

These envelopes are piecewise linear curves that specify the upper and lower limits on a signal at all instants of simulation time. The SIM Optimal Design VI then calculates $H(p)$ as the minimum and maximum distance of the time series points from these envelopes.

Use the **Inequality Constraints** parameter of the SIM Optimal Design VI to define these envelopes. This parameter specifies the upper and lower constraint envelopes on four areas of the dynamic system: the control action, the output, the rate of change of the control action, and the rate of change of the output.

Note You can use the Graphically Specify Inequality Constraints VI, located in the `labview\examples\simulation\Optimization Based Design\Graphically Specify Inequality Constraints\` directory, to draw the upper and lower envelopes. This VI returns a set of points you then can wire to the **Inequality Constraints** parameter.

For example, consider an output $\boldsymbol{y}_i(t)$ constrained by envelopes as shown in Figure 6-3.

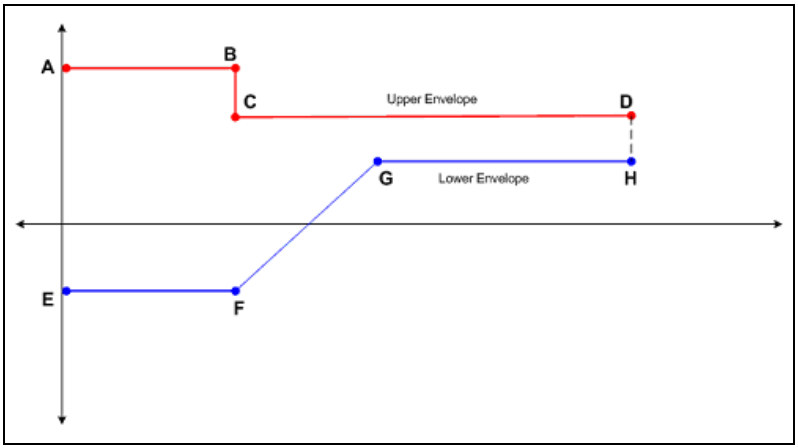

Figure 6-3. Upper and Lower Constraint Envelopes

A, B, C, and D are points that define the upper envelope $UE_i(t)$, and E, F, G, and H are points that define the lower envelope $LE_i(t)$. The SIM Optimal Design VI then constrains $\boldsymbol{y}_i(t)$ to the following relationship:

$$LE_i(t) < \boldsymbol{y}_i(t) < UE_i(t)$$

This VI encodes this constraint by computing the clearance between the output and each envelope. The upper clearance UC_i is defined as $max\ (UE_i(t) - \boldsymbol{y}_i(t))$. The lower clearance LC_i is defined as $max\ (\boldsymbol{y}_i(t) - LE_i(t))$. These clearances further clarify the constraints, as the following relationships show:

$$-\varepsilon < UC_i < \infty$$
$$-\varepsilon < LC_i < \infty$$

where $\varepsilon = 1e^{-21}$.

You also can place constraints on the rate of change of control actions and outputs. If at least five points are available, this VI computes these rates of change using the following equation:

$$\dot{f}(t) = \frac{f(t-2h) - 8f(t-h) + 8f(t+h) - f(t+2h)}{12h}$$

where t is the simulation time, h is the space between time steps, and $f(t)$ is an output or control action signal.

At boundaries, or if fewer than five points are available, this VI uses the following equations instead:

$$\dot{f}(t) = \frac{f(t+h) - f(t-h)}{2h} \qquad or \qquad \dot{f}(t) = \frac{f(t+h) - f(t)}{h}$$

You can use a VI template to specify custom calculations for implementing the inequality constraints. To load this template, in the **New** dialog box, select **VI»From Template»Simulation»Optimization Based Design» SIM Compute Inequality Constraints.vit** from the **Create New** tree. To see an example of how to define and manipulate these parameters, open the SIM Compute Inequality Constraints (Default) VI, located in the `labview\vi.lib\addons\simulation\Optimization Based Design\Constraints\` directory. This VI implements the inequality constraints the previous equations specified.

After you save the custom inequality constraint calculations as a VI, you must specify the location of the custom function in the **Problem Specification** parameter of the SIM Optimal Design VI. Select **User defined** for the **Inequality constraints type** parameter and specify the path to the VI in the **File path user defined inequality constraints** path control.

Defining Parameter Bounds

Parameter bounds are constraints on parameter values being optimized. For example, while searching for the best value of the spring constant, you might know that springs are available only in a certain range. In this case, you can specify the parameter k must stay within minimum and maximum values. Parameter bounds are important because these bounds define the parameter space in which the SQP algorithm searches for optimal values.

Use the **Parameter Bounds** parameter of the SIM Optimal Design VI to specify minimum and maximum values for each parameter.

Defining Initial Parameter Values and a Mesh

After you define the parameter space using the minimum and maximum values of each parameter, you must specify the initial values of each parameter. These initial parameter values determine where the SQP algorithm begins the search for optimal values. However, if you choose only a single initial set of initial values, the SQP algorithm might return local optimal values. Local optimal values are values that minimize the cost function within only a subset of parameter space. Local optimal values are not the true solution to the SQP algorithm because the true optimal values might exist outside the parameter space the algorithm searched.

To mitigate this problem, you can execute the SQP algorithm several times, using a different set of initial parameter values each time. If you use a large enough range of initial parameter values within the given parameter space, you can be relatively confident that the SQP algorithm finds the global optimal values.

You can implement this strategy by defining an initial parameters mesh. The initial parameters mesh defines the distribution pattern of these sets of initial values and the total number of initial value sets to generate. You can choose from four patterns depending on the needs of the problem: **Uniform grid**, **Uniform random**, **Quasirandom**, and **Random walk**.

Each pattern has unique characteristics and strengths. For example, the simplest possible option is the uniform grid, which generates a specified number of equally-spaced locations in the parameter space. However, the uniform random and quasirandom options often provide better coverage of the parameter space while using a fewer number of points than the uniform grid option. The random walk option biases the search to explore close to the initial values but eventually explores a larger region of parameter space. This option is useful if you think a particular parameter space contains the optimal values and you want to focus on a certain region of that space, such as the center.

Use the **Initial Parameters** parameter of the SIM Optimal Design VI to specify initial parameter values. Use the **Initial Parameters Mesh** parameter of this VI to define an initial parameters mesh.

Executing the SQP Algorithm

The SIM Optimal Design VI uses an internal simulation diagram to obtain the finite-horizon time-domain response of the dynamic system model. Use the **Solver Parameters** parameter of this VI to configure the simulation. You also can configure the SQP algorithm using the **beginning state**, **cno settings**, and **stopping criteria** parameters.

This VI returns the following information:

- **Optimal parameters**—A list of possible optimal parameter values. Each column of this array corresponds to one parameter you specified in the **Parameter Bounds** array. Each row of this array corresponds to one execution of the SQP algorithm.
- **Optimal costs**—The results of the specified cost function that correspond to each row of the **Optimal parameters** array.
- **Design parameters**—The set of parameter values that minimize the specified cost function. These values are the optimal parameter values.
- **Design cost**—The result of the specified cost function if you apply the values from the **Design parameters** array.
- **Signals**—The finite-horizon time-response data for the output and control action of the dynamic system, evaluated at each point specified in the **Optimal parameters** array.

The SQP algorithm takes as long to execute as the product of the number of function evaluations and the run time of the simulation. If you specify only one set of initial parameter values, the algorithm must solve, on average, between 30 and 200 functions. The front panel of the SIM Optimal Design VI includes a **Current Data** page that you can use to monitor the progress of the algorithm as the VI runs. This page updates each time the SQP algorithm executes from one set of initial parameter values.

The **Optimal Design Parameters** page of this VI also includes the **Best Parameters (Infeasible Constraints)** and **Best cost (Infeasible Constraints)** parameters. These parameters return optimal parameter values and the associated cost function result with no constraints. This information can be useful when revising the constraint envelopes.

If the dynamic system has constraints and the SQP algorithm does not return feasible optimal values, try ensuring that the specified cost function remains constant when parameter values are outside the feasible range. This method helps you set reasonable parameter bounds. Additionally, reducing system discontinuities helps the SQP algorithm execute precisely.

You can use several methods to reduce discontinuities, for example, avoiding saturation effects and rate limiters in the system model. Refer to the *Simulation Discontinuities* section of Chapter 5, *Solving Ordinary Differential Equations*, for information about discontinuities.

Case Study: Designing a PID Controller for a Second-Order System

This section examines the PID Design for Second Order Continuous System VI, which determines the optimal gain values K_P, K_I, and K_D for a PID controller in a second-order continuous dynamic system. This VI is located in the `labview\examples\simulation\Optimization Based Design\` directory. Figure 6-4 shows the block diagram of this VI.

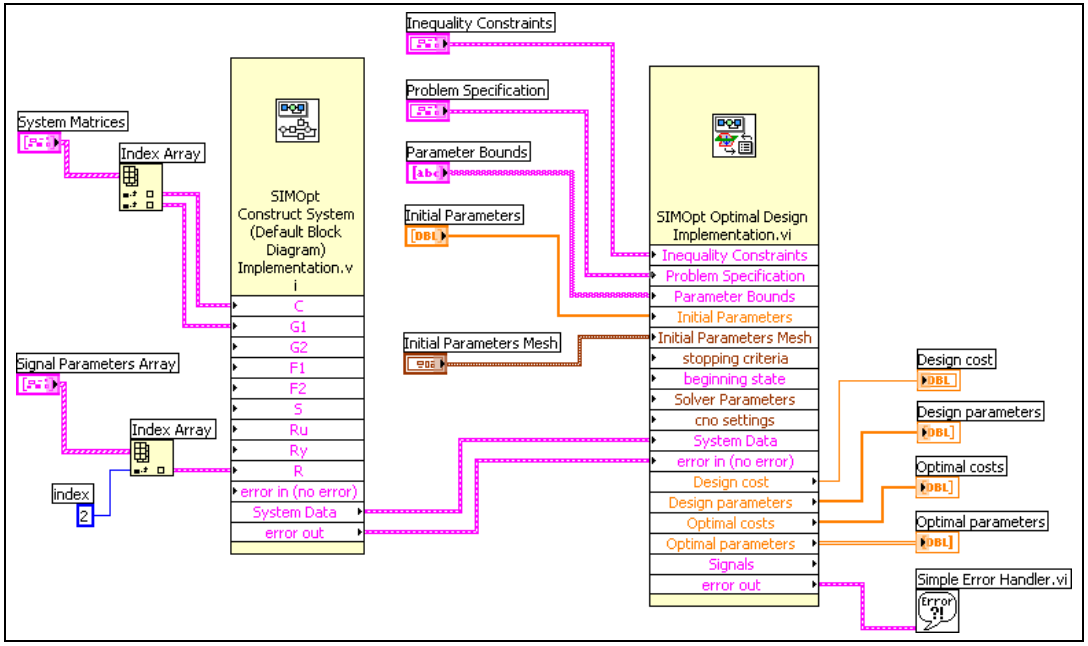

Figure 6-4. Block Diagram of the PID Design for Second Order Continuous System VI

This VI constructs a dynamic system model using the SIM Construct Default System VI. Refer to the *Constructing the Dynamic System Model* section of this chapter for the structure of this dynamic system. In this example, the dynamic system has only three components: a controller C, a plant G1, and a reference input signal ***r***. The SIM Optimal Design VI excites the dynamic system with the reference input signal to determine the optimal settings for the controller.

The following transfer function equation defines the plant G1.

$$G_1(s) = \frac{1}{s^2 + 4s + 2}$$

Figure 6-5 shows the **System Matrices** control that represents this equation.

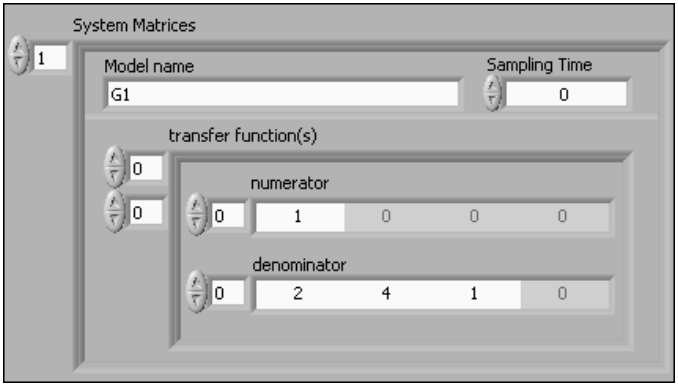

Figure 6-5. Defining the Plant G1

The **Signal Parameters Array** parameter of the SIM Construct Default System VI specifies the reference input signal ***r*** that excites this system. This example excites the dynamic system with a step input signal. Figure 6-6 shows this parameter.

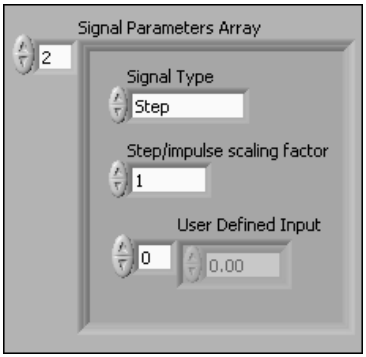

Figure 6-6. Exciting the Controller with a Step Input Signal

Note Notice in Figure 6-6 that ***r*** is the third element, or index number 2, of the **Signal Parameters Array**. r_u and r_y correspond to the first and second elements of this array, respectively. If you define a custom dynamic system, you also can define custom reference signals beginning with the third element of this array.

Figure 6-7 shows the **Problem Specification** parameter of the SIM Optimal Design VI.

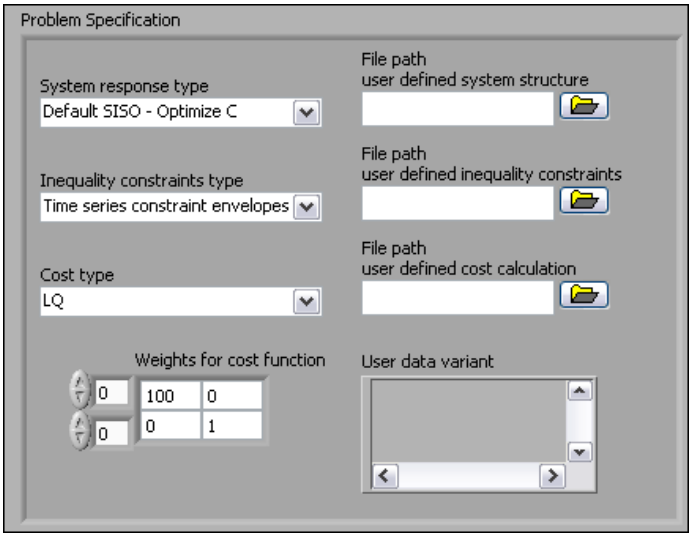

Figure 6-7. Specifying the Problem this Example Solves

The **System response type** parameter in Figure 6-7 shows this example optimizes parameters for the controller C. The **Inequality constraints type** parameter shows this example uses the default calculations for the inequality constraint envelopes. The **Cost type** parameter shows this example minimizes the linear quadratic (**LQ**) cost function. The following equation shows this cost function.

$$J_{LQ} = \int_0^T e^T(t)\boldsymbol{W}u(t)dt \cong \sum_{n=0}^{N} \Delta t \cdot e^T(n)\boldsymbol{W}u(n)$$

The **Weights for cost function** parameter in Figure 6-7 shows this example uses the following weight matrix ***W***:

$$\boldsymbol{W} = \begin{bmatrix} 100 & 0 \\ 0 & 1 \end{bmatrix}$$

This weight matrix penalizes the control action but emphasizes that this example minimizes the position error. ***W*** also reflects the difference in scale between the control action range and the output range.

Figure 6-8 shows the **Inequality Constraints** parameter that defines the inequality constraints envelopes.

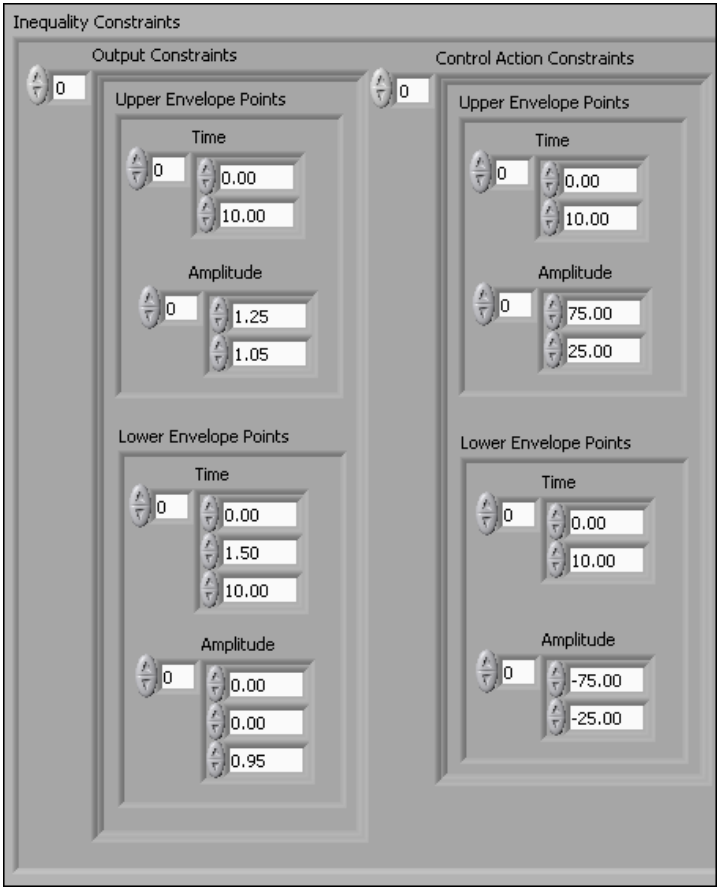

Figure 6-8. Specifying Inequality Constraints on the Output and Control Action

The upper constraint envelope on the output is a line with points (0, 1.25) and (10, 1.05). The lower constraint envelope on the output has points (0, 0), (0, 1.5), and (10, 0.95). Figure 6-9 shows these envelopes.

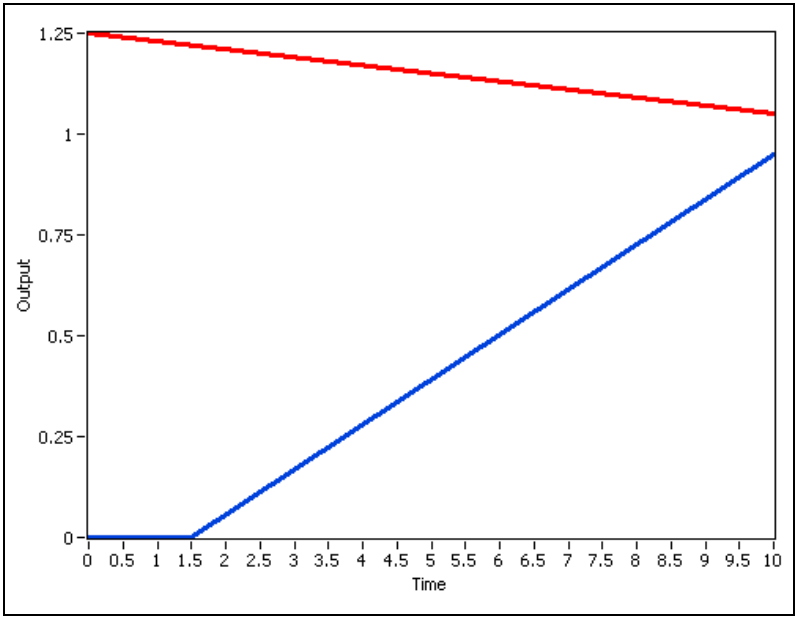

Figure 6-9. Constraints on the Output

Figure 6-8 also shows the constraints on the control action. The upper constraint envelope is a line with points (0, 75) and (10, 20). The lower

constraint envelope is a line with points (0, –75) and (10, –25). Figure 6-10 shows these envelopes.

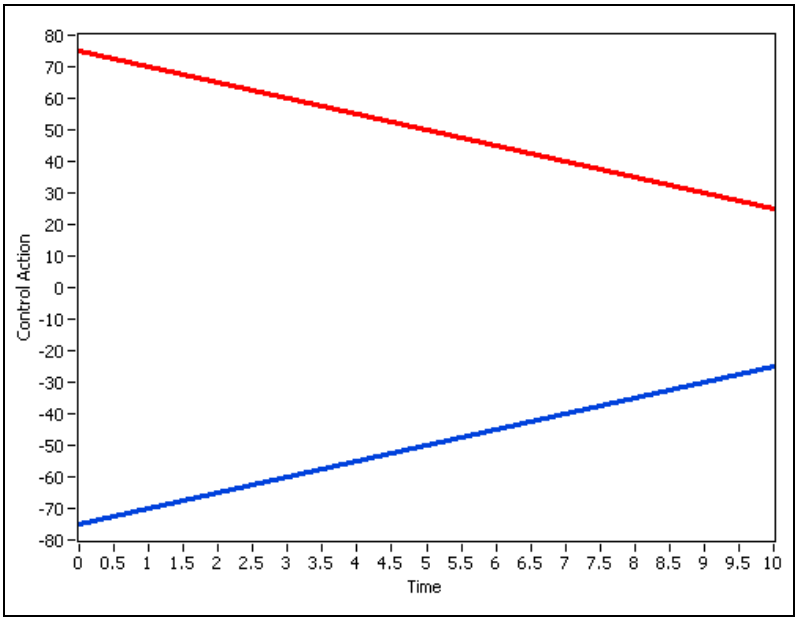

Figure 6-10. Constraints on the Control Action

This example does not place inequality constraints on either the rate of control action or the rate of output.

Figure 6-11 shows the parameter bounds this example uses for each gain parameter of the PID controller.

Parameter Bounds

	Min	Max
Parameter 1 (P)	0	50
Parameter 2 (I)	0	25
Parameter 3 (D)	0	15

Figure 6-11. Specifying Parameter Bounds for the Gain Parameters of the PID Controller

These bounds form the parameter space in which this VI searches for optimal values.

Figure 6-12 shows the initial parameter values this example uses. Each element of this array corresponds to same element of the **Parameter Bounds** array.

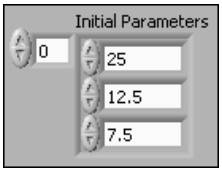

Figure 6-12. Initializing the Values of the PID Controller Parameters

This example uses an initial parameters mesh to generate two additional search locations quasirandomly. Figure 6-13 shows this **Initial Parameters Mesh**.

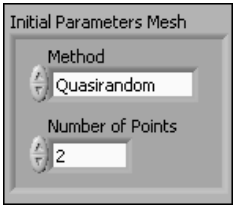

Figure 6-13. Specifying the Initial Parameters Mesh

When you run this VI, the SQP algorithm executes once using the values from the **Initial Parameters** parameter. The algorithm then executes two more times, using the **Initial Parameters Mesh** parameter to generate two sets of initial parameter values.

Figure 6-14 shows all three sets of initial parameter values.

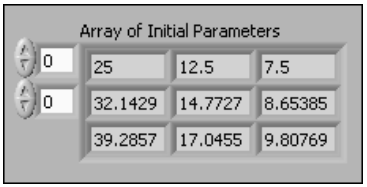

Figure 6-14. All Sets of Initial Parameter Values

The first row of Figure 6-14 is the same as the values shown in Figure 6-12. The second and third rows contain quasirandomly-generated locations within the **Parameter Bounds** of each parameter.

Note The **Array of Initial Parameters** parameter is on the **Debug Information** page of the SIM Optimal Design VI.

After the SQP algorithm executes, this VI returns all possible optimal parameter values, as shown in Figure 6-15.

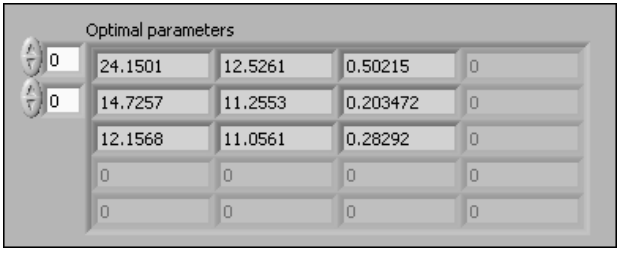

Figure 6-15. Possible Sets of Optimal Parameter Values

Of the parameter value sets shown in Figure 6-15, one set minimizes the LQ cost function. The SIM Optimal Design VI returns this optimal set of

parameter values in the **Design parameters** array. Figure 6-16 shows this array and the corresponding value of the cost function.

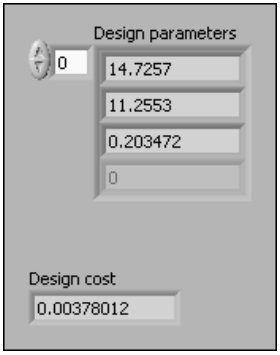

Figure 6-16. Optimal Parameters and Cost Function Value for the Controller of this Dynamic System

Therefore, the optimal gain values for K_P, K_I, and K_D are 14.7257, 11.2553, and 0.203472, respectively.

7

Using the Simulation Translator

You can use the Simulation Translator to convert a `.mdl` file, developed in The MathWorks, Inc. Simulink® simulation environment, into a LabVIEW VI that consists of a simulation diagram containing LabVIEW functions, wires, and simulation subsystems corresponding to the contents of the `.mdl` file.

Note The Simulation Translator cannot convert diagrams developed with The MathWorks, Inc. Stateflow® application software or other Simulink blocksets.

This chapter provides information about using the Simulation Translator to convert models developed in the Simulink simulation environment into LabVIEW code.

Converting Models into LabVIEW Code

Select **Tools»Control Design and Simulation»Simulation Translator** from the pull-down menu to launch the **Simulation Translator** dialog box. Refer to the *Simulation Translator Dialog Box* topic of the *LabVIEW Help,* available by selecting **Help»Search the LabVIEW Help**, for information about specific Simulation Translator options.

The Simulation Translator executes the following steps:

1. Executes any `.m` files that the `.mdl` file includes in addition to any `.m` files you specify in the dialog box. This step obtains the values of any parameters and equations that the `.m` file contains. However, the Simulation Translator executes this step only if The MathWorks, Inc. MATLAB® software is installed on the same computer as the LabVIEW Simulation Module.
2. Parses the `.mdl` file for model information such as timing and ordinary differential equation (ODE) solver settings.
3. Stores each system, subsystem, block, and line in an XML-based Common Graph Description (CGD) format.
4. Generates a LabVIEW simulation subsystem corresponding to each model subsystem, in order. Converting in order ensures that the Simulation Translator generates every LabVIEW subsystem that has a

parent simulation diagram before generating the parent simulation diagram. The Simulation Translator converts all blocks into one or more LabVIEW functions or simulation subsystems. The Simulation Translator also converts lines into wires that connect terminals on the LabVIEW functions and simulation subsystems.

Common Warnings

If the Simulation Translator cannot find a value for a parameter in the `.mdl` file it is converting, LabVIEW displays a warning. In these cases, the Simulation Translator uses the default value of the parameter in the corresponding LabVIEW function.

Note In some cases, the Simulation Translator cannot find a value for a parameter because the parameter contains an expression instead of a constant value. If the MATLAB software is installed on the computer, the Simulation Translator attempts to evaluate the MATLAB software expressions in the `.mdl` file prior to converting the file. If the Simulation Translator successfully evaluates the expression, the Simulation Translator uses the result of that evaluation as the parameter value and does not produce a warning.

The Simulation Translator cannot fully convert all functions of every model to LabVIEW block diagram code. If the Simulation Translator encounters a block it cannot convert, you receive a warning. In these cases, the Simulation Translator creates a placeholder simulation subsystem. You must create a simulation subsystem using a LabVIEW VI to accomplish the same functionality as the block to replace this placeholder simulation subsystem. Refer to the *Unsupported Blocks* topic of the *LabVIEW Help* for a list of the blocks the Simulation Translator cannot convert.

Because LabVIEW is strict about data types, the converted simulation subsystem might have broken wires. In this case, add block diagram code to convert between converted data types.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

B

BDF	Backwards difference formula. Also known as Gear's Method.

C

CGD	Common Graph Description. The format the Simulation Translator uses to store each system, subsystem, block, and line from a model developed in The MathWorks, Inc. Simulink® simulation environment.
continuous model	Dynamic system model used to represent real-world signals that vary continuously with time. A continuous model is characterized by differential equations.
controller	Device that regulates the operation of a dynamic system.

D

direct feedthrough	Relationship between a function input and a function output in which the function uses the input at the current step to calculate the output at the current step.
discrete model	Dynamic system model used to represent signals that are sampled in time at discrete intervals. A discrete model is characterized by difference equations.
distributed parameter model	Physical model that can be described by partial differential equations.
dynamic system	System whose behavior varies with time.

E

empirical modeling	Modeling technique in which you use experimental data to define a system model.

F

feedback cycle	Cycle in which data flow originates from an output of a function or subsystem and terminates as an input of the same function or subsystem.

G

global error	Maximum difference between the solution the function computes and the exact solution.

H

HIL	Hardware-in-the-loop. A simulation configuration in which you test a controller implementation with a software model of the plant.

I

indirect feedthrough	Relationship between a function input and a function output in which the function does not use the input at the current step to compute the output at the current step.
Input Node	A collection of input terminals attached to the Simulation Loop. Use the Input Node to configure simulation parameters programmatically.

L

linear model	Model that obeys the principle of superposition.
linearize	A procedure that approximates the behavior of a nonlinear model.
local error	Error introduced into the solution per time step.
lumped parameter model	Physical model described by an ordinary differential equation.

M

major time step	A time step evaluated at time $t + dt$.
minor time step	A time step evaluated between major time steps.
multi-step ODE solver	ODE solver that approximates the behavior of a model at time $t + dt$ by taking into account the behavior of the model at a number of previous time steps.
model	Set of differential or difference equations that represent the behavior of a controller, simulation, or dynamic system.

N

nonlinear model	Model that does not obey the principle of superposition.

O

ODE	Ordinary differential equation.
order	ODE solver characteristic that determines how much the error amount changes when you vary the step size.
Output Node	An output terminal on the Simulation Loop. Use the Output Node to view any errors the simulation diagram generates.

P

PECE	Predict-evaluate-correct-evaluate.
period	The amount of time in which a discrete Simulation function must complete.
physical modeling	Modeling technique in which you use the laws of physics to define a system model.
plant	Physical system whose behavior you want to observe, replicate, or manipulate.

R

RCP	Rapid control prototype. A simulation configuration in which you test plant hardware with a software model of the controller.

S

simulation diagram	LabVIEW diagram that allows you to use Simulation functions within a Simulation Loop or simulation subsystem. A simulation diagram, like other LabVIEW diagrams, has the following semantic properties: • The order of operations is not completely specified by the user. • The order of operations is implied by data interdependencies. • A function may only execute after all necessary inputs have become available. • Outputs are generated after a function completes execution.
Simulation Loop	Loop that executes the simulation diagram over multiple time steps.
single-step ODE solver	ODE solver that approximates the behavior of a model at time $t + dt$ by taking into account only the behavior of the model at time t.
skew	The amount of time by which you want to delay the execution of a discrete Simulation function.
step size	Size of the interval of one time step.
stiff ODE solver	ODE solver used to evaluate a stiff model.
stiff system	System whose dynamics are described by widely varying time constants.
subsystem	A section of a simulation diagram you represent with a single icon instead of multiple Simulation functions and wires.

T

time step	Interval from t to $t + dt$.
time-invariant model	Model whose parameters do not change with time.

time-variant model	Model whose parameters change with time.
trim	A procedure that searches for the values of states and inputs that produce output and/or state derivative conditions you specify.

V

variable step-size ODE solver	ODE solver that adjusts the step size throughout the simulation to ensure that the per-step error remains at a given relative and absolute tolerance.