Simulink® Control Design

For Use with Simulink®

- Modeling
- Simulation
- Implementation

Advanced Topics

Version 1

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Simulink Control Design Advanced Topics

Revision History

June 2004	Online only	New for Version 1.0 (Release 14)
October 2004	Online only	Revised for Version 1.1 (Release 14SP1)
March 2005	Online only	Revised for Version 1.2 (Release 14SP2)
September 2005	Online only	Revised for Version 1.3 (Release 14SP3)

Contents

1 Understanding and Controlling Results

1

Understanding and Controlling Results

To create accurate linearized models, it is important to be able to interpret the results and to understand the linearization algorithms. One method of interpreting the results is by simulating the linearized model and comparing the output with the original model. The linearization algorithms can be adjusted in various ways to control these results, as outlined in this chapter.

Comparing the Linearized and Original Models

Comparing simulations of the original model with simulations of the linearized model helps to determine if the linearized system behaves in a similar way to the original model. To make this comparison, re-insert the linearized subsystem into the model, configure the inputs and operating points so that they are the same as in the original model, and then compare output signals from a simulation of the two models.

When comparing models, remember that the states, inputs, and outputs of the linearized model are defined about an operating point of the original model, using the following variables:

$$\delta x(t) = x(t) - x_0$$
$$\delta u(t) = u(t) - u_0$$
$$\delta y(t) = y(t) - y_0$$

This means that when the original model is at the operating point $x(t)=x_0$, $u(t)=u_0$, $y(t)=y_0$, the linearized model will be at the operating point $\delta x(t)=0$, $\delta u(t)=0$, $\delta y(t)=0$. To compare the models accurately, subtract u_0 from input values and x_0 from the initial state values in the linearized model, then add y_0 to the output signal.

When you linearize only a portion of the original model, you should simulate the linearized model by substituting it back into the model in place of the original portion. This ensures that the operating point and inputs to the linearized portion are correct. To do this, export the linearized model to the workspace, delete the original portion from the model, and replace it with an LTI System block based on the linearized model.

Example

This example compares the `magball` model with the linearized model computed in “Linearizing the Model” in the online documentation:

1. If you have not done so already, linearize the `magball` model at the targeted operating point computed in “Computing Operating Points from Specifications”.

2. To create a new model containing the linearized plant system, first export the linearized model and operating point from the Control and Estimation Tools Manager to the MATLAB® workspace. To do this, right click the linearized model name in the project tree of the Control and Estimation Tools Manager. Select **Export** from the menu. Accept the default name for the model, `Model_sys`, and for the operating point, `Model_op`.

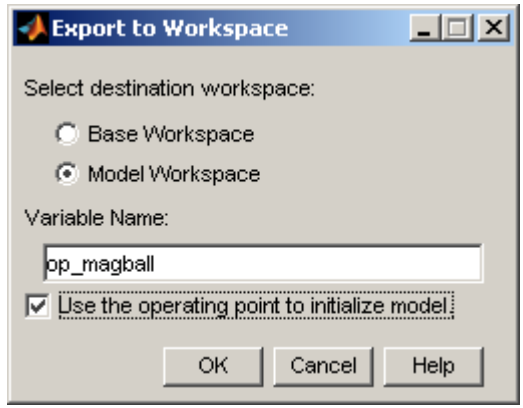
3. Create a new Simulink® model, `magball_lin`, which is a copy of the original model, `magball`. Replace the Magnetic Ball Plant subsystem in `magball_lin` with an LTI System block (located in the Control System Toolbox category of the Simulink Library Browser). Import the linearized model into this block by entering `Model_sys` in the **LTI system variable** field in the Block Parameters window.

4. For simulations of the nonlinear and linearized models to be compared, you need to set the operating points for each model by specifying the initial values of the states in the models:

   a. `magball`

      To set the initial values for the `magball` model, in the Control and Estimation Tools Manager, right click on the operating point that you used for the linearization, and select **Export to Workspace** to open the Export to Workspace dialog box.

      Within the Export to Workspace dialog box, select **Model Workspace** as the location to export the operating point to, and select the check box labeled **Use the operating point to initialize model**.



Click **OK** to export the operating point to the model workspace and use it to define the initial values of states in the model.

b `magball_lin`

In `magball_lin`, the operating point values for the linearized system will all be zero since this subsystem was linearized about the operating point values. The operating point values in the Controller will be the same as in the original model since the Controller was not linearized. To create a vector of initial state values with the correct state ordering, first create a new operating point object for the system by typing

```
op=operpoint('magball_lin')
```

Change the operating point for the Controller in `op` to be the same as those in `Model_op`.

```
op.States(1).x=Model_op.States(1).x
```

This returns the following operating point.

```
Operating Point for the Model magball_lin.
(Time-Varying Components Evaluated at time t=0)

States:
----------
(1.) magball_lin/Controller/Controller
      x: 0
```

```
      x: -2.56e-006
(2.) magball_lin/LTI System/Internal
      x: 0
      x: 0
      x: 0

Inputs: None
```

Keep the operating point for the LTI system as zero.

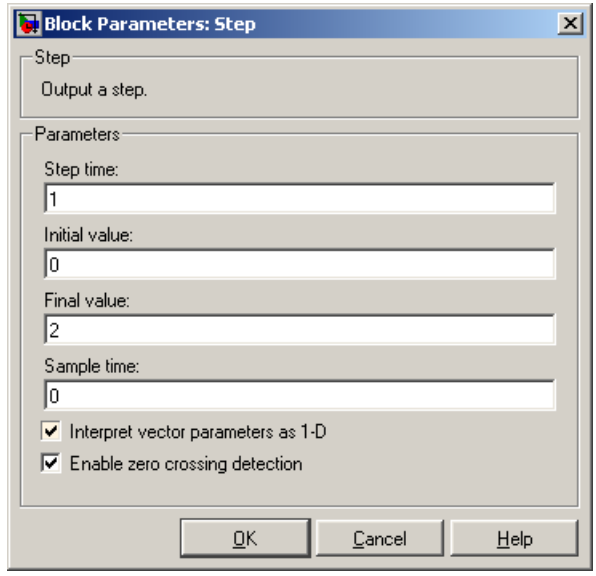
Create a Simulink structure from this operating point using the `getstatestruct` function. The structure contains the operating point values in a format that Simulink can use to set initial values.

```
x_struct=getstatestruct(op);
```

To use the values in `x_struct1`, as initial values for `magball_lin`, select **Simulation > Configuration Parameters** in the `magball_lin` model window, then select the **Data Import/Export** panel. Select the check box next to **Initial State** and enter `x_struct` on the right. Click **OK** .

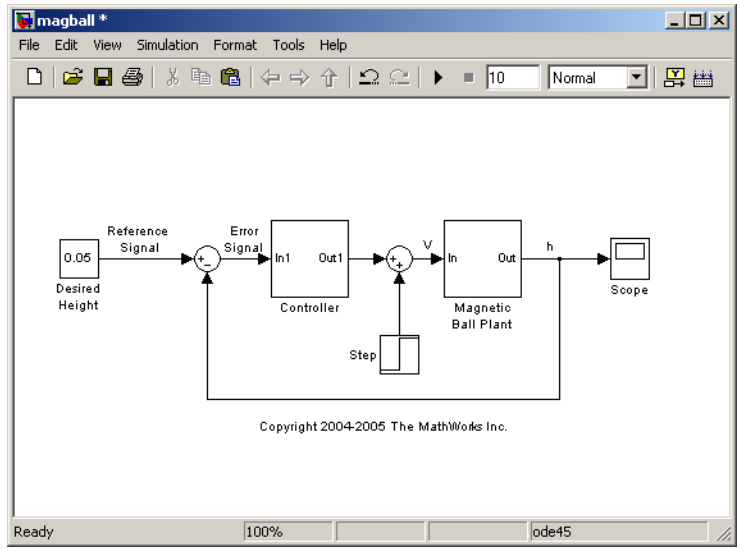
5. The output of `magball_lin` will be zero at the operating point. To create an output signal that is comparable with that in `magball`, add a Constant block, with a value of `0.05` to the output of `magball_lin`. Similarly, the input to `magball_lin` should be zero at the operating point. This is achieved by subtracting a value of 14 from the input signal of the linearized system. The operating point values, 0.05 and 14, were found using a Scope block to measure steady-state signal levels in the original model.

6. To observe the response of the models to a perturbation, add a Step block with the following parameter values to the input to the plant in both models.

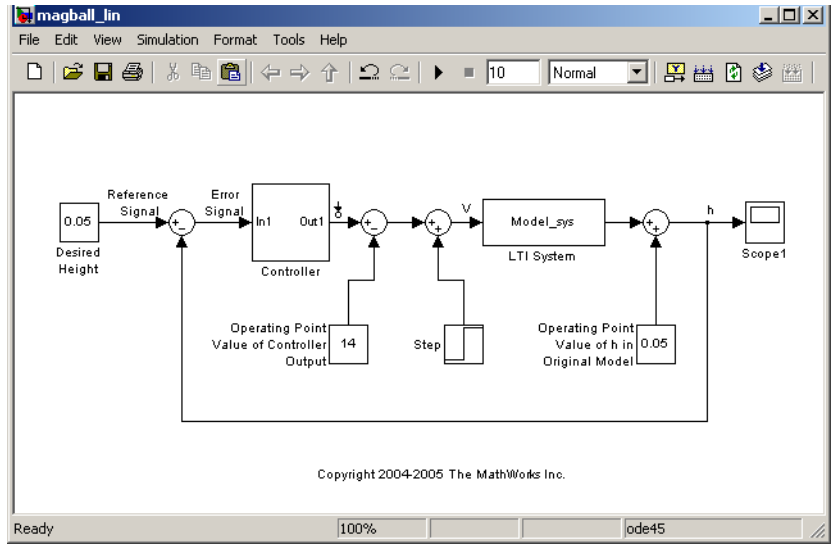


Parameter Values for Step Block

The model diagrams should now look like those in the following figures.

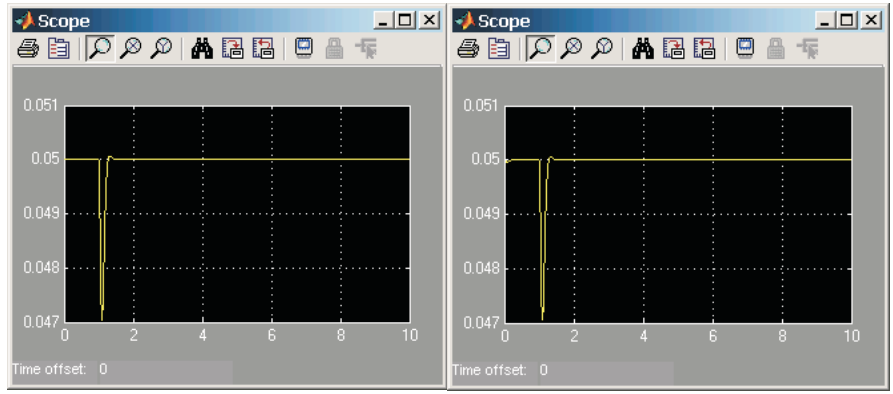


Magball Model with a Step Block Added to the Input



Magball Model with Linearized Magnetic Ball Plant

7 Run simulations in both models. The output signals, in the Scope blocks, are shown in the following figure.



Scope Blocks from Original (left) and Linearized (right) Models

As shown in the figure, both the original and linearized models react to the step input in a similar way.

Linearization Algorithms

Simulink Control Design can use two different linearization methods. The default method, which is used unless an option is selected, is called block-by-block analytic linearization. To use the alternative method, numerical-perturbation linearization, you must select an option in the Linearization Options dialog box of the GUI, or if using functions, with the `linoptions` function. The remainder of this chapter describes the two linearization methods in more detail and provides suggestions for controlling the results to create more accurate linearized models.

The default linearization method, block-by-block analytic linearization, linearizes the blocks individually and then combines the results to produce the linearization of the whole system. This method has several advantages:

- It divides the linearization problem into several smaller, easier problems.
- It defines the system being linearized by input and output markers on the signal lines rather than root-level inport and outport blocks.
- It supports open loop analysis.
- You can control the linearization of each block by using an analytic linearization that is programmed into the block or by selecting a perturbation level for the block.

The main disadvantage of this method is that it cannot be used with models that contain model references using the Model block.

Alternatively, numerical-perturbation linearization linearizes the *whole system* by numerically perturbing the system's inputs and states about the operating point. This method has the advantage that it is quick and simple, especially for large or complicated systems. In addition it is the only linearization algorithm that supports models containing model references. However, there are also several disadvantages with this method:

- It relies on root-level inport and outport blocks to define the system being linearized.
- There is no support for open loop analysis.
- You have limited control over the perturbation levels for each block.

- It does not use any of the analytic, pre-programmed block linearizations.
- It is sensitive to scaling issues (models with large and small signal values).

"Block-by-Block Analytic Linearization" on page 1-11 and "Numerical-Perturbation Linearization" on page 1-27 discuss these methods further.

Block-by-Block Analytic Linearization

Block-by-block analytic linearization is the default linearization method in Simulink Control Design. In this method, each of the blocks within the linearization path is first linearized individually. The linearization of the whole system is then computed by combining these results using the algorithm discussed in . This approach breaks the problem into several smaller problems. The following section gives details of the methods used to linearize each block, with suggestions for controlling the linearizations to create more accurate linearized models.

Note The block-by-block analytic linearization algorithm does not work with models that contain references to other models using the Model block. Use the numerical perturbation linearization algorithm to linearize these models.

Individual Block Linearization Methods

There are two methods that Simulink Control Design uses to linearize the individual blocks in a model. Each method has options that you can control to create accurate linearized models.

Analytic Linearization

Many Simulink blocks contain analytic Jacobians for exact linearization. When linearizing a system using block-by-block analytic linearization, you can use these analytic linearizations instead of numerically perturbing the block. This is especially useful for blocks that contain discontinuities and do not give good results using numerical perturbation.

The following table lists the Simulink blocks that contain analytic Jacobians for linearization. For more information see the reference page for each block.

Analytic Block Jacobians

Block	Analytic Jacobian (Y/N)	Notes
Continuous Library		
Derivative	Y	Allows control of the time constant for the filter constant
Integrator	Y	Includes option to exclude saturation and resets from linearization
State-Space	Y	
Transfer Fcn	Y	
Transport Delay	Y	Allows control of Padé order
Variable Transport Delay	Y	Allows control of Padé order
Zero-Pole	Y	
Discontinuities Library		
Backlash	N	
Coulomb and Viscous Friction	N	
Dead Zone	Y	Includes option to treat as gain when linearizing
Dead Zone Dynamic	Y	
Hit Crossing	N	
Quantizer	Y	Includes option to treat as gain when linearizing
Rate Limiter	Y	Includes option to treat as gain when linearizing
Rate Limiter Dynamic	N	
Relay	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Saturation	Y	Includes option to treat as gain when linearizing
Saturation Dynamic	N	
Wrap to Zero	N	
Discrete Library		
Difference	Y	
Discrete Derivative	N	
Discrete Filter	Y	
Discrete State-Space	Y	
Discrete Transfer Fcn	Y	
Discrete Zero-Pole	Y	
Discrete-Time Integrator	Y	Includes option to ignore saturation and resets during linearization. Jacobian not supported for non-double data types.
First-Order Hold	N	
Integer Delay	N	
Memory	Y	Linearizes to a gain of 1 when driven by a continuous signal, linearizes to a Unit Delay when driven by a discrete signal.
Tapped Delay	N	
Transfer Fcn First Order	Y	
Transfer Fcn Lead or Lag	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Transfer Fcn Real Zero	Y	
Unit Delay	Y	Jacobian does not support frame-based signals
Weighted Moving Average	N	
Zero-Order Hold	N	
Logic and Bit Operations Library		
Bit Clear	N	
Bit Set	N	
Bitwise Operator	N	
Combinatorial Logic	N	
Compare To Constant	N	
Compare To Zero	N	
Detect Change	N	
Detect Decrease	N	
Detect Fall Negative	N	
Detect Fall Nonpositive	N	
Detect Increase	N	
Detect Rise Nonnegative	N	
Detect Rise Positive	N	
Extract Bits	Y	
Interval Test	N	
Interval Test Dynamic	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Logical Operator	N	
Relational Operator	N	
Shift Arithmetic	N	
Lookup Tables Library		
Cosine	N	
Direct Lookup Table (n-D)	N	
Interpolation (n-D) using PreLookup	Y	
Lookup Table	N	
Lookup Table (2-D)	N	
Lookup Table (n-D)	N	
Lookup Table Dynamic	N	
PreLookup Index Search	Y	
Sine	N	
Math Operations Library		
Abs	Y	
Add	Y	
Algebraic Constraint	N	
Assignment	N	
Bias	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Complex to Magnitude-Angle	N	
Complex to Real-Imag	N	
Divide	Y	
Dot Product	N	
Gain	Y	
Magnitude-Angle to Complex	N	
Math Function	N	
Matrix Concatenation	N	
MinMax	N	
MinMax Running Resettable	N	
Polynomial	N	
Product	Y	
Product of Elements	Y	
Real-Imag to Complex	N	
Reshape	N	
Rounding Function	N	
Sign	Y	Linearizes to `Inf` at zero, linearizes to zero otherwise
Sine Wave Function	N	
Slider Gain	Y	
Subtract	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Sum	Y	
Sum of Elements	Y	
Trigonometric Function	N	
Unary Minus	N	
Weighted Sample Time Math	N	
Model Verification Library		
Assertion	N/A	Does not contain outputs
Check Discrete Gradient	N/A	Does not contain outputs
Check Dynamic Gap	N/A	Does not contain outputs
Check Dynamic Lower Bound	N/A	Does not contain outputs
Check Dynamic Range	N/A	Does not contain outputs
Check Dynamic Upper Bound	N/A	Does not contain outputs
Check Input Resolution	N/A	Does not contain outputs
Check Static Gap	N/A	Does not contain outputs
Check Static Lower Bound	N/A	Does not contain outputs
Check Static Range	N/A	Does not contain outputs
Check Static Upper Bound	N/A	Does not contain outputs
Model Wide Utilities Library		

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Block Support Table	N/A	Does not contain outputs
DocBlock	N/A	Does not contain outputs
Model Info	N/A	Does not contain outputs
Time-Based Linearization	N/A	Does not contain outputs
Trigger-Based Linearization	N/A	Does not contain outputs
Ports and Subsystems Library		
Configurable Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Atomic Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
CodeReuse Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Enable	N	
Enabled and Triggered Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Enabled Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
For Iterator Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Function-Call Generator	N/A	Only the blocks within the subsystem are part of the linearization
Function-Call Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
If	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
If Action Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Inport	N/A	Does not contain outputs
Model	N	
Outport	N/A	Does not contain inputs
Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Switch Case	N	
Switch Case Action Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Trigger	N	
Triggered Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
While Iterator Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Signal Attributes Library		
Data Type Conversion	Y	
Data Type Conversion Inherited	Y	
Data Type Duplicate	N/A	Does not contain outputs
Data Type Propagation	N/A	Does not contain outputs
Data Type Scaling Strip	Y	
IC	N	
Probe	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Rate Transition	Y	
Signal Conversion	Y	
Signal Specification	Y	
Weighted Sample Time	N	
Width	N	
Signal Routing Library		
Bus Assignment	Y	
Bus Creator	Y	
Bus Selector	Y	
Data Store Memory	N/A	Does not contain inputs or outputs
Data Store Read	Y	Linearizes to a gain of 1. Assumes that there is no delay between data store read and data store write.
Data Store Write	Y	Linearizes to a gain of 1. Assumes that there is no delay between data store read and data store write.
Demux	N/A	
Environment Controller	Y	
From	N/A	
Goto	N/A	
Goto Tag Visibility	N/A	
Index Vector	Y	
Manual Switch	Y	
Merge	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Multiport Switch	Y	
Mux	N/A	
Selector	Y	
Switch	Y	
Sources Library - N/A No Inputs		
Sinks Library - N/A No Outputs		
User Defined Functions Library		
Embedded MATLAB Function	N	
Fcn	N	
Level-2 M-File S-Function	N	
MATLAB Fcn	N	
S-Function	N	
S-Function Builder	N	
Additional Math and Discrete Library		
Fixed-Point State-Space	Y	
Transfer Fcn Direct Form II	N	
Transfer Fcn Direct Form II Time Varying	N	
Unit Delay Enabled	Y	
Unit Delay Enabled External IC	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Unit Delay Enabled Resettable	Y	
Unit Delay Enabled Resettable External IC	Y	
Unit Delay External IC	Y	
Unit Delay Resettable	Y	
Unit Delay Resettable External IC	Y	
Unit Delay With Preview Enabled	Y	
Unit Delay With Preview Enabled Resettable	Y	
Unit Delay With Preview Enabled Resettable External RV	Y	
Unit Delay With Preview Resettable	Y	
Unit Delay With Preview Resettable External RV	Y	
Decrement Real World	Y	
Decrement Stored Integer	Y	
Decrement Time To Zero	Y	
Decrement To Zero	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Increment Real World	Y	
Increment Stored Integer	Y	

Several of these blocks include options to control the linearization that you can adjust in the Block Parameters window. For example, you can change the order of the Padé approximation used in the Transport Delay block or select the **Treat as gain when linearizing** option in the Saturation block. The **Notes** column of the table above gives details on blocks that include these options.

Note The preprogrammed, analytic block linearizations are only used in block-by-block analytic linearization. When using numerical-perturbation linearization, these blocks will be numerically perturbed along with the rest of the system.

Block Perturbation

When a preprogrammed block linearization cannot be used, Simulink Control Design will compute the block linearization by numerically perturbing the states and inputs of the block about the operating point of the block. As opposed to the numerical-perturbation linearization method, this perturbation is local and its propagation through the rest of the model is restricted.

The block perturbation algorithm involves introducing a small perturbation to the nonlinear block and measuring the response to this perturbation. Both the perturbation and the response are used to create the matrices in the linear state-space model of this block. Changing the size of the perturbations will change the resulting linearized model.

As described in “Linearization of Nonlinear Models”, a nonlinear Simulink block can be written as a state-space system:

$$\dot{x}(t) = f\left(x(t), u(t), t\right)$$
$$y(t) = g\left(x(t), u(t), t\right)$$

In these equations, $x(t)$ represents the states of the block, $u(t)$ represents the inputs of the block, and $y(t)$ represents the outputs of the block.

A linearized model of this system is valid in a small region around the operating point $t=t_0$, $x(t_0)=x_0$, $u(t_0)=u_0$, and $y(t_0)=g(x_0,u_0,t_0)=y_0$. Subtracting the operating point values from the states, inputs, and outputs defines a set of variables centered about the operating point:

$$\delta x(t) = x(t) - x_0$$
$$\delta u(t) = u(t) - u_0$$
$$\delta y(t) = y(t) - y_0$$

The linearized model can be written in terms of these new variables and is usually valid when these variables are small, i.e. when the departure from the operating point is small:

$$\delta\dot{x}(t) = A\delta x(t) + B\delta u(t)$$
$$\delta y(t) = C\delta x(t) + D\delta u(t)$$

The state-space matrices A, B, C, and D of this linearized model represent the Jacobians of the block, as defined in "Linearization of Nonlinear Models". To compute the matrices, the states and inputs are perturbed, one at a time, and the response of the system to this perturbation is measured by computing $\delta\dot{x}$ and δy. The perturbation and response are then used to compute the matrices in the following way

$$A(:,i) = \frac{\dot{x}\big|_{x_{p,i}} - \dot{x}_o}{x_{p,i} - x_o}, \qquad B(:,i) = \frac{\dot{x}\big|_{u_{p,i}} - \dot{x}_o}{u_{p,i} - u_o}$$

$$C(:,i) = \frac{y\big|_{x_{p,i}} - y_o}{x_{p,i} - x_o}, \qquad D(:,i) = \frac{y\big|_{u_{p,i}} - y_o}{u_{p,i} - u_o}$$

where

- $x_{p,i}$ is the state vector whose ith component is perturbed from the operating point value.
- x_o is the state vector at the operating point.
- $u_{p,i}$ is the input vector whose ith component is perturbed from the operating point value.
- u_o is the input vector at the operating point.
- $\dot{x}|_{x_{p,i}}$ is the value of $\dot{x}$ at $x_{p,i}$, u_o.
- $\dot{x}|_{u_{p,i}}$ is the value of $\dot{x}$ at $u_{p,i}$, x_o.
- $\dot{x}_o$ is the value of $\dot{x}$ at the operating point.
- $y|_{x_{p,i}}$ is the value of y at $x_{p,i}$, u_o.
- $y|_{u_{p,i}}$ is the value of y at $u_{p,i}$, x_o.
- y_o is the value of y at the operating point.

Linearized models of discrete-time or multi-rate blocks are computed in a similar way. See "Linearization of Discrete-Time Models" and "Linearization of Multi-Rate Models" for the equations of linearized discrete-time and multi-rate systems.

Note A perturbed value is one that has been changed by a very small amount from the operating point value. The default difference between the perturbed value and the operating point value is $10^{-5}(1+|x|)$ for block-by-block analytic linearization, where x is the operating point value.

Changing the size of the perturbations will change the linearization results.

The default perturbation size is $10^{-5}(1+|x|)$ where x is the operating point value of the state or input being perturbed. To change the perturbation

size of the states in the Magnetic Ball Plant block in the `magball` model to $10^{-7}(1+|x|)$, type

```
blockname='magball/Magnetic Ball Plant'
set_param(blockname,'StatePerturbationForJacobian','1e-7')
```

To change the perturbation size of the input of the Magnetic Ball Plant block to $10^{-7}(1+|u|)$, where u is the input signal level, follow these steps:

1 Get the block's port handles

```
ph=get_param('magball/Magnetic Ball Plant','PortHandles')
```

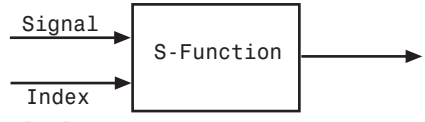
2 Get the inport

```
pin=ph.Inport(1)
```

3 Set the perturbation level for this inport

```
set_param(pin,'PerturbationForJacobian','1e-7')
```

If there is more than one inport, you can choose to assign a different perturbation level to each. The following figure shows an S-Function block with two input signals, the actual signal and an index variable. Since you probably do not want to perturb the index signal, you can assign a perturbation level of zero to this inport.



Block Containing Two Inports

Numerical-Perturbation Linearization

An alternative linearization method available for use in Simulink Control Design is numerical-perturbation linearization, which computes state-space matrices for the linearized model by numerical perturbation of the *whole system*. The method is relatively quick and simple, although as mentioned in "Linearization Algorithms" on page 1-9, it does have some disadvantages.

Numerical-perturbation linearization requires that root-level inport and outport blocks be present in the model. These blocks define the portion of the model that you want to linearize instead of inserting input and output points by right-clicking on the signal lines. Any input, output, or open loop points on signal lines in the model will be ignored when using numerical-perturbation linearization.

The perturbation is introduced to the system at the root level inport blocks and in the states of the system. The response to the perturbation is measured at the outport blocks.Suggestions for controlling the results of numerical-perturbation linearization to create accurate linearized models are given in "Controlling the Results of Numerical-Perturbation Linearization" on page 1-30

Note The numerical perturbation linearization algorithm is the only linearization algorithm that works with models that contain references to other models using the Model block.

Invoking Numerical-Perturbation Linearization

Prior to Simulink 3.0, numerical-perturbation linearization was the only linearization method available with Simulink. Although block-by-block analytic linearization is now the default linearization method, you might choose to use numerical-perturbation linearization if your model is very large or complicated.

To use numerical-perturbation linearization with the Simulink Control Design GUI, select **Tools > Options** while in the **Linearization Task** node of the Control and Estimation Tools Manager and select `Numerical-Perturbation` from the **Linearization Algorithms** menu.

To use numerical-perturbation linearization with the `linearize` function, set the `LinearizationAlgorithm` option to `'numericalpert'` with the `linoptions` function.

```
linopt=linoptions('LinearizationAlgorithm','numericalpert')
```

To linearize the model, type

```
sys=linearize('modelname',op,linopt)
```

where `modelname` is the name of the model being linearized and `op` is the operating point object for the system.

Perturbation Algorithm

The numerical perturbation algorithm involves introducing a small perturbation to the nonlinear model and measuring the response to this perturbation. Both the perturbation and the response are used to create the matrices in the linear state-space model. Changing the size of the perturbations will change the resulting linearized model.

As described in "Linearization of Nonlinear Models", a nonlinear Simulink model can be written as a state-space system:

$$\dot{x}(t) = f\left(x(t)u(t),t\right)$$
$$y(t) = g\left(x(t)u(t),t\right)$$

In these equations, $x(t)$ represents the states of the model, $u(t)$ represents the inputs of the model, and $y(t)$ represents the outputs of the model.

A linearized model of this system is valid in a small region around the operating point $t=t_0$, $x(t_0)=x_0$, $u(t_0)=u_0$, and $y(t_0)=g(x_0,u_0,t_0)=y_0$. Subtracting the operating point values from the states, inputs, and outputs defines a set of variables centered about the operating point:

$$\delta x(t) = x(t) - x_o$$
$$\delta u(t) = u(t) - u_o$$
$$\delta y(t) = y(t) - y_o$$

The linearized model can be written in terms of these new variables and is usually valid when these variables are small, i.e. when the departure from the operating point is small:

$$\delta\dot{x}(t) = A\delta x(t) + B\delta u(t)$$
$$\delta y(t) = C\delta x(t) + D\delta u(t)$$

The state-space matrices A, B, C, and D of this linearized model represent the Jacobians of the system, as defined in "Linearization of Nonlinear Models". To compute the matrices, the states and inputs are perturbed, one at a time, and the response of the system to this perturbation is measured by computing $\delta\dot{x}$ and δy. The perturbation and response are then used to compute the matrices in the following way

$$A(:,i) = \frac{\dot{x}|_{x_{p,i}} - \dot{x}_o}{x_{p,i} - x_o}, \qquad B(:,i) = \frac{\dot{x}|_{u_{p,i}} - \dot{x}_o}{u_{p,i} - u_o}$$

$$C(:,i) = \frac{y|_{x_{p,i}} - y_o}{x_{p,i} - x_o}, \qquad D(:,i) = \frac{y|_{u_{p,i}} - y_o}{u_{p,i} - u_o}$$

where

- $x_{p,i}$ is the state vector whose ith component is perturbed from the operating point value.
- x_o is the state vector at the operating point.
- $u_{p,i}$ is the input vector whose ith component is perturbed from the operating point value.
- u_o is the input vector at the operating point.
- $\dot{x}|_{x_{p,i}}$ is the value of $\dot{x}$ at $x_{p,i}$, u_o.
- $\dot{x}|_{u_{p,i}}$ is the value of $\dot{x}$ at $u_{p,i}$, x_o.
- $\dot{x}_o$ is the value of $\dot{x}$ at the operating point.

- $y|_{x_{p,i}}$ is the value of y at $x_{\mathrm{p,i}}$, u_{o}.
- $y|_{u_{p,i}}$ is the value of y at $u_{\mathrm{p,i}}$, x_{o}.
- y_{o} is the value of y at the operating point.

Linearized models of discrete-time or multi-rate systems are computed in a similar way. See "Linearization of Discrete-Time Models" and "Linearization of Multi-Rate Models" for the equations of linearized discrete-time and multi-rate systems.

Note A perturbed value is one that has been changed by a very small amount from the operating point value. The default difference between the perturbed value and the operating point value is $10^{-5} + 10^{-8}|x|$ for numerical-perturbation linearization.

Controlling the Results of Numerical-Perturbation Linearization

Several factors influence the creation of accurate linearized models. "What Is Linearization?" discusses some of these factors, such as careful selection of operating points. Factors that are particular to numerical-perturbation linearization are presented here, with suggestions for controlling them.

Setting the Perturbation Level

In numerical-perturbation linearization, there are three options for setting the perturbation levels of states and inport blocks:

1. You can accept the default perturbation levels. The default perturbation levels for the states are $10^{-5} + 10^{-8}|x|$, where x is a Simulink structure or vector of the operating point values for the states in the model. Similarly, default perturbation levels for the inport blocks are $10^{-5} + 10^{-8}|u|$, where u is a Simulink structure or vector of the operating point values for the inputs in the model.

2 You can edit the linearization property `NumericalPertRel` using the `linoptions` function. The value of this property adjusts the perturbations in the following way:

- The perturbation of the states is

  $$\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times |x|.$$

- The perturbation of the inputs is

  $$\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times |u|.$$

When using the Control and Estimation Tools Manager graphical interface, select **Tools > Options** to open the Options dialog, and then select the **Linearization** tab-panel. Within the **Linearization** panel, make sure that you have selected `Numerical perturbation` as the **Linearization algorithm** and then enter a value for **Relative Perturbation level** under **Options for numerical perturbation algorithm**.

3 You can provide individual perturbation levels for each state and inport block. These values override the values computed using the `NumericalPertRel value`. Set the perturbation levels using the `linoptions` function to edit the linearization properties `NumericalXPert` and `NumericalUPert`. To specify the absolute perturbation levels for `NumericalXPert` and `NumericalUPert`, you can use the `operpoint` function to create an operating point object and then edit the operating point values using dot-notation or the `set` function.

When using the Control and Estimation Tools Manager graphical interface, select **Tools > Options** to open the Options dialog, and then select the **Linearization** tab-panel. Within the **Linearization** panel, make sure that you have selected `Numerical perturbation` as the **Linearization algorithm** and then enter values for **State Perturbation level** and **Input Perturbation level** under **Options for numerical perturbation algorithm**. You can enter either scalars or operating point objects created with the `operpoint` function. **State Perturbation level** and **Input Perturbation level** values override **Relative Perturbation level** values.

Example: Command-Line Numerical Perturbation Linearization of a Model Reference Model

Simulink Control Design supports linearization of models that contain references to other models, using the Model block, however you must use the Numerical Perturbation linearization algorithm to linearize these models. The following example illustrates how to linearize a model reference model at the MATLAB command line using numerical perturbation. For information on linearizing a model reference model using the Simulink Control Design graphical interface, see "Example: Using the Graphical Interface for Numerical Perturbation Linearization of a Model Reference Model" on page 1-34.

1 Open the model.

In this example, we will use the `scdairframe_reference.mdl` model, included with Simulink Control Design. The model uses a Model block to reference another Simulink model, `eom.mdl`, hence numerical perturbation is the only linearization algorithm that you can use with this model.

Enter

```
scdairframe_reference
```

at the MATLAB command line to open this model.

2 Set Inport and Outport blocks.

Linearization using the numerical perturbation algorithm is between the root level Inport and Outport blocks, rather than input and output points on signal lines. When your model does not already contain Inport or Outport blocks, you need to add them to the points where you want to perturb the model, and measure the response. Note that in this case the `scdairframe_reference` model already contains one Inport block and two Outport blocks.

3 Create an operating point object for the model.

There are several possible methods for doing this, depending on the model you are using and the information you have about the operating point. See "Specifying Operating Points Using Functions" in the online documentation for more information on creating operating points. For the purposes of this

example, to illustrate the use of the numerical perturbation linearization algorithm, we will simply create a default operating point with the following command:

```
op_point=operpoint('scdairframe_reference')
```

4 Specify the linearization algorithm

By default, the linearization algorithm is set to block-by-block linearization. To change the algorithm to numerical perturbation you need to create a linearization options object and set the `'LinearizationAlgorithm'` field to `'numericalpert'`, using the following command:

```
options=linoptions('LinearizationAlgorithm','numericalpert')
```

5 Set the perturbation levels

By default, the state and input perturbation levels are set to

$$1e^{-5} + 1e^{-8}|x|$$

and

$$1e^{-5} + 1e^{-8}|u|$$

respectively, where $|x|$ and $|u|$ are the absolute values of the states and inputs. These values should be sufficient for most applications and you should not typically need to change them. However, if you want to specify individual perturbation values for each state, you can create an operating point object, edit the state values within this object, and then assign, these values to the `NumericalXPert` option, using the following commands:

```
state_pert=operpoint('scdairframe_reference');
state_pert.states(1).x=[1e-8;1e-9];
state_pert.states(2).x=1e-7;
state_pert.states(3).x=[1e-7;1e-8];
state_pert.states(4).x=1e-9;
options.NumericalXPert=state_pert;
```

6 Linearize model

The following command linearizes the model about the chosen operating point, using the perturbation settings in the linearization options object, and returns the state-space matrices of the linearized model:

```
sys=linearize('scdairframe_reference',op_point,options)
```

Example: Using the Graphical Interface for Numerical Perturbation Linearization of a Model Reference Model

In the previous example, a model reference model, `scdairframe_reference.mdl`, was linearized using Simulink Control Design functions for numerical perturbation. In the following example, we will use the numerical perturbation algorithm to linearize the same model within the Control and Estimation Tools Manager graphical interface.

1 Open the model.

In this example, we will use the `scdairframe_reference.mdl` model, included with Simulink Control Design. The model uses a Model block to reference another Simulink model, `eom.mdl`, hence numerical perturbation is the only linearization algorithm that you can use with this model.

Enter

```
scdairframe_reference
```

at the MATLAB command line to open this model.

2 Set Inport and Outport blocks.

Linearization using the numerical perturbation algorithm relies on perturbing root level Inport and Outport blocks, rather than input and output points on signal lines. When your model does not already contain Inport or Outport blocks, you need to add them to the points where you want to perturb the model, and measure the response. Note that in this case the `scdairframe_reference` model already contains one Inport block and two Outport blocks.

3 Open a linearization task for the model in the Control and Estimation Tools Manager.

Within the `scdairframe_reference.mdl` model window, select **Tools > Control Design > Linear Analysis**. This opens the Control and Estimation Tools Manager and creates a task for linearization. Note that the Control and Estimation Tools Manager displays a warning dialog to inform you that it has automatically selected the numerical perturbation linearization algorithm for the model. Click the **OK** button the close this dialog.

You should also notice that since you will numerically perturb this model using root-level Inport and Outport blocks, you cannot specify any linearization points in the **Analysis I/Os** panel of the **Linearization Task**.

4 Create an operating point object for the model.

There are several possible methods for doing this, depending on the model you are using and the information you have about the operating point. See "Specifying Operating Points" in the online documentation for more information on creating operating points. For the purposes of this example, to illustrate the use of the numerical perturbation linearization algorithm, we will skip this step and use the default operating point for the linearization.

5 Specify the linearization algorithm

Since you can only linearize the `scdairframe_reference.mdl` model using the numerical perturbation algorithm, the Control and Estimation Tools Manager selected this algorithm automatically when the linearization task was created. To select numerical perturbation linearization as the algorithm for a model that does not use model references, select **Tools > Options** within the Control and Estimation Tools Manager to open the Options dialog, select the **Linearization** panel in the Options dialog, and then select `Numerical perturbation` as the **Linearization algorithm**.

6 Set the perturbation levels

To use perturbation levels other than the default settings, select **Tools > Options** within the Control and Estimation Tools Manager to open the Options dialog, and then select the **Linearization** panel. Under **Options for numerical perturbation algorithm**, enter perturbation

values. The perturbation values can be either scalars, vectors, operating point objects, or Simulink structures of state values.

For this example, enter `1e-9` in the **State perturbation level** box. This value overrides the state perturbation values computed from the **Relative perturbation level** setting. However, because we have not explicitly specified the **Input perturbation level**, the algorithm will still use the **Relative perturbation level** setting to compute input perturbations.

Note that these perturbation values are not the same as the perturbation values used in the previous example.

7 Linearize the model

a Select **Linearization Task** in the panel on the left of the Control and Estimation Tools Manager.

b Select the **Operating Points** panel on the right.

c Within the **Operating Points** panel, select the operating point that you want to use for the linearization. For this example, there should be only one choice, the default operating point.

d Click the **Linearize Model** button to linearize the model around this operating point. The results are plotted in the LTI Viewer.

Handling Special Blocks

Certain blocks, especially those containing discontinuities such as `Saturation` or `Transport Delay`, may not linearize well using numerical-perturbation. Although these blocks often have preprogrammed linearizations that are used with block-by-block analytic linearization instead of numerically perturbing them, they are *not* used in numerical-perturbation linearization. An alternative solution is to replace these blocks with an appropriate block before linearizing your model. For example, you might choose to replace a `Saturation` block with a `Gain` block.

Random Number blocks inside models that reference other models using the Model block, can also sometimes cause inaccurate numerical perturbation linearization results. Care should be taken when linearizing or computing operating points with model reference models that use these blocks.

Handling Feedback Loops

"Understanding Open Loop Analysis" in the online documentation discusses the effect of feedback loops on the results of a linearization. With block-by-block analytic linearization, you can perform open loop analysis without removing feedback loops. When using numerical-perturbation linearization, the only way to remove the effect of feedback loops is to manually remove them from the model *and* manually force the operating point to remain the same as the original model.

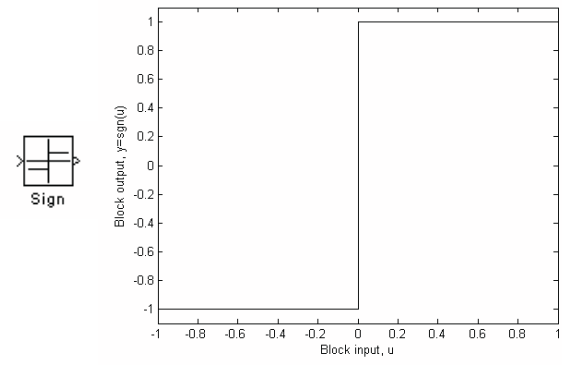
Recommendations for Computing Operating Points and Creating Accurate Linearized Models

Particular blocks and modeling situations in Simulink can sometimes cause difficulties with computing operating points (trimming) and linearization. However, by understanding what it means to trim or linearize a Simulink model and by using the correct modeling techniques, you can create accurate operating points and linearized models for use in further analysis and design.

This section consists of examples that highlight modeling situations that can lead to problems when computing operating points and linearized models, with recommendations for ways to avoid these situations. The examples focus on the following modeling situations:

Blocks with Discontinuities

There are several Simulink blocks that contain discontinuities, such as the Sign block, whose behavior is shown in the following figure.



The very large derivatives that occur at the point of discontinuity can cause problems with linearization. For example, the Sign block has the following linearization

$$D = 0, u \neq 0$$

$$D = \infty, u = 0$$

where D is a state-space matrix, and u is the input signal to the block.

When these blocks are within the linearization path of your model, the resulting linearized model could potentially have very large values. There is no obvious solution to this problem and it is recommended that you remove or replace these blocks. However, when your model operates in a region away from the point of discontinuity, the linearization will be zero. This should not cause any problems, although when the linearizations of several blocks are multiplied together (as in a feedback path) it can cause the linearization of the system to be zero.

When these blocks are outside the linearization path, they can still contribute to the definition of the operating point of the model but will not otherwise affect the linearization. It is safe to use them for reference signals, disturbances, and any other signals and blocks that are not being linearized.

Other examples of blocks with discontinuities include

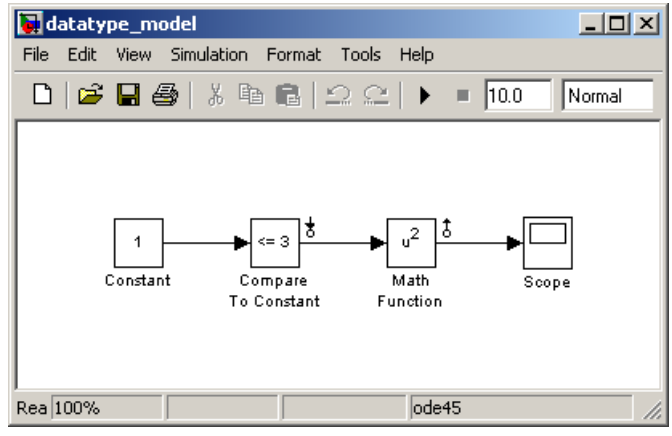
- Relational Operator blocks

- Relay block
- Logical Operator blocks
- Stateflow blocks
- Quantizer block (has an option to treat as a gain when linearizing)
- Saturation block (has an option to treat as a gain when linearizing)
- Deadzone block (has an option to treat as a gain when linearizing)

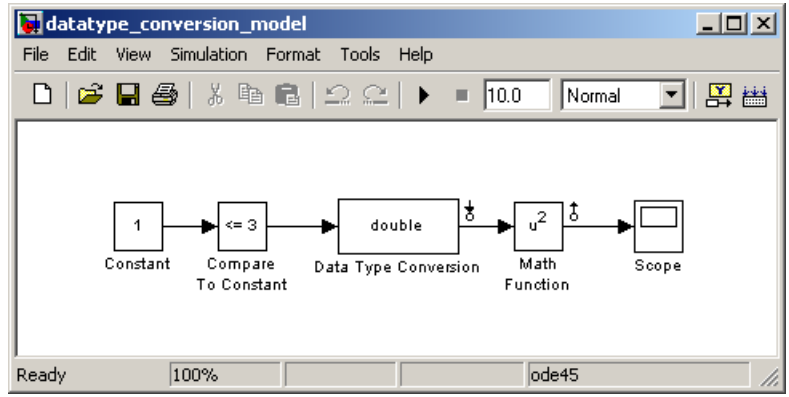
Non-Double Data Types

Blocks that have non-double data type signals as either inputs or outputs, and which do not have a preprogrammed exact linearization, will automatically linearize to zero as they cannot be numerically perturbed. For example, many logical operator blocks have Boolean outputs and will therefore linearize to zero.

To work around this problem, you can use a Data Type Conversion block, which does have a preprogrammed exact linearization, to convert your signals to doubles before linearizing the model. The following example illustrates this concept. The model in this example is configured to linearize the Square block at an operating point where the input is 1. The resulting linearized model should be 2 but the input to the Square block is Boolean and the linearization is zero.



However, by inserting a Data Type Conversion block before the linearization input point, the input signal to the Square block is a double, and the linearized model gives the correct response of 2.



Overriding Non-Double Data Types

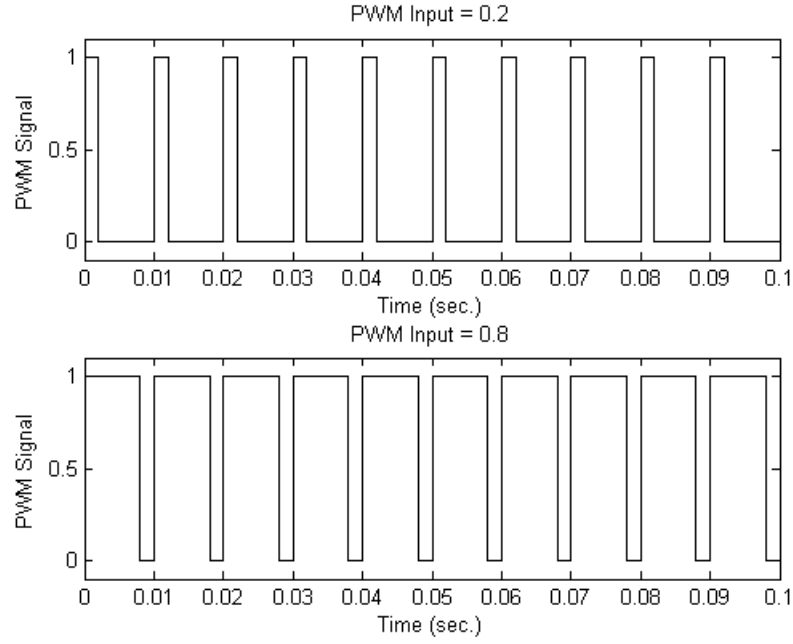
When linearizing a model that contains non-double data types but still runs correctly in all double precision, you can choose to override all data types with doubles. To do this, in the model window select **Tools > Fixed-Point Settings** from the menu. This opens the Fixed-Point Settings window. Within this window select **True doubles** from the **Data type override** menu. When linearizing and simulating the model, it now uses doubles for all data types.

Note This method does not work when the model relies on other data types in its algorithm, such as relying on integer data types to perform truncation from floats.

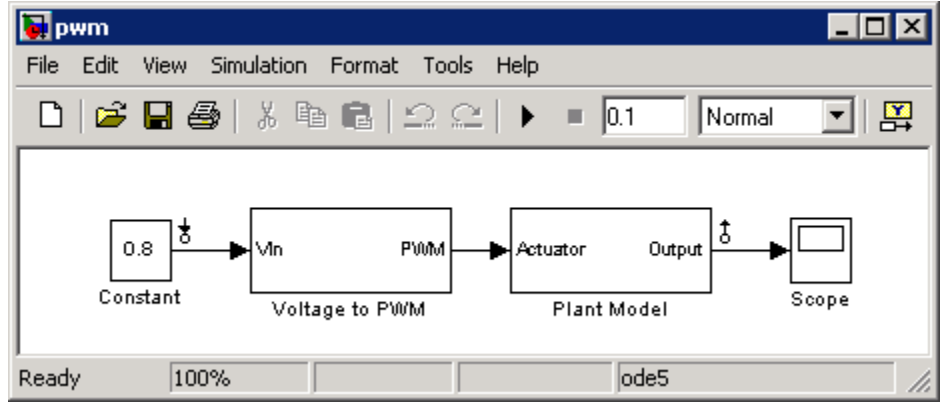
Pulse Width Modulation

Many industrial applications use Pulse Width Modulation (PWM) signals because of their robustness in the presence of noise. The following figure shows two examples of PWM signals. In the first example, a DC voltage of 0.2V is represented by a PWM signal with a 20% duty cycle (a value of 1 for 20% of the cycle, followed by a value of 0 for 80% of the cycle). The average

signal value is 0.2V. The second example shows a PWM representation of a 0.8V DC signal, where the duty cycle is 80%.

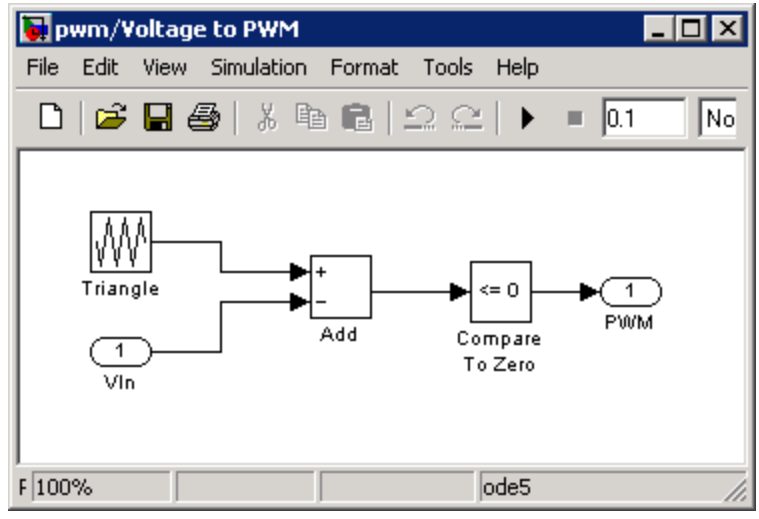


The model, `pwm.mdl`, shown below, converts a constant signal to a PWM signal.



When linearizing a model containing PWM signals there are two effects you should take into account:

- The signal level at the operating point will be one of the discrete values within the PWM signal, not the DC signal value. For example, in the model above, the signal level will be either 0 or 1, not 0.8. This change in operating point will affect the linearized model.
- The creation of the PWM signal within the subsystem `Voltage to PWM`, shown below, uses a comparator block, the Compare to Zero block. Comparator blocks do not linearize well due to their discontinuities and the non-double outputs.



A solution to the two problems described above is to consider removing the PWM block before linearizing the model.

Transport Delay, Memory, and Other Blocks with Non-Trimmable States

Blocks with non-double, discrete states cannot accurately be used to compute operating points from design specifications (also called trimming) because these special states cannot be seen by the `findop` function. Blocks that contain these states include

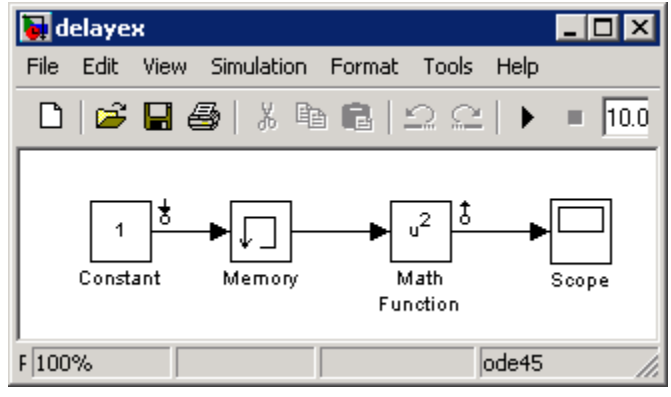
- Action Subsystem blocks which are not enabled
- Backlash block
- Embedded MATLAB Function block with persistent data
- Transport Delay and Variable Transport Delay blocks
- Memory block
- Rate Transition block
- Stateflow blocks
- S-Function blocks with states not registered as Continuous or Double Value Discrete

To determine when your model contains any of these blocks with states that cannot be trimmed, run the following command

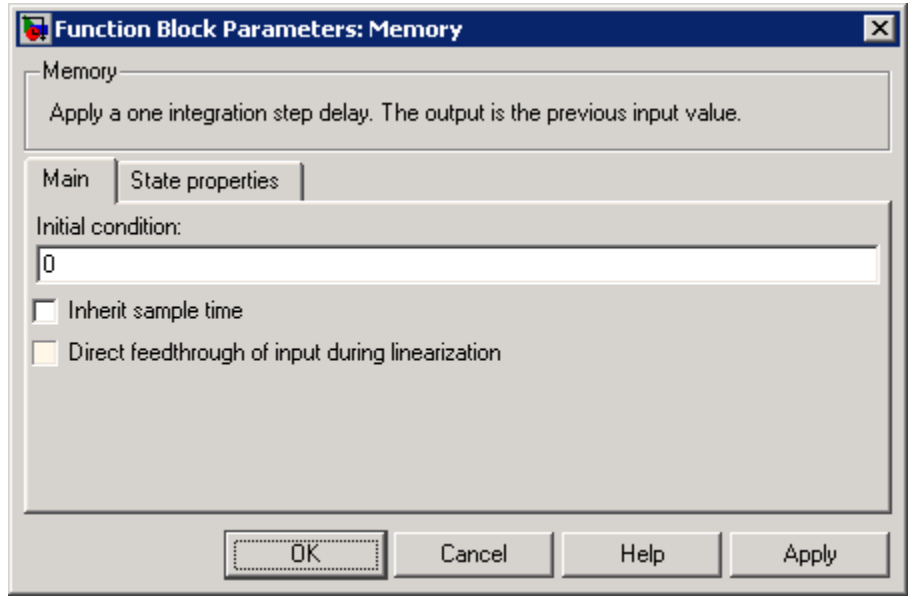
```
sldiagnostics('modelname','CountBlocks')
```

which returns a list of all the blocks in the model and the number of occurrences of each.

When you have Memory blocks or Variable Transport Delay/Transport Delay blocks in your model, you can properly configure the initial outputs of these blocks so that trim or linearization uses the correct output value. The model `delayex.mdl`, shown below, illustrates this issue.



In this model the Memory block is configured in the block dialog to have an initial output of 0 but is driven by a Constant block with an output of 1. This causes the output signal of the block to be 0 in the operating point. However, in the steady-state operating point for this model, the output of the Memory block is 1. To create an accurate operating point or a linearized model that is based on this more accurate operating point, select the **Direct feedthrough of input during linearization** option in the block dialog. This will force the output of the Memory block to be the same as the input during trim or linearization.



The problem of block output during trim or linearization also occurs for the Backlash block although, in this case, the block does not have a direct feedthrough option. Extra care should be taken when linearizing a model containing Backlash blocks.

For other blocks with states that are not seen by trim, you should consider removing them from the model while trimming. If the output of the block does not effect any of the state derivatives or desired output levels downstream it does not pose a problem for trim and you do not need to remove it. If the block

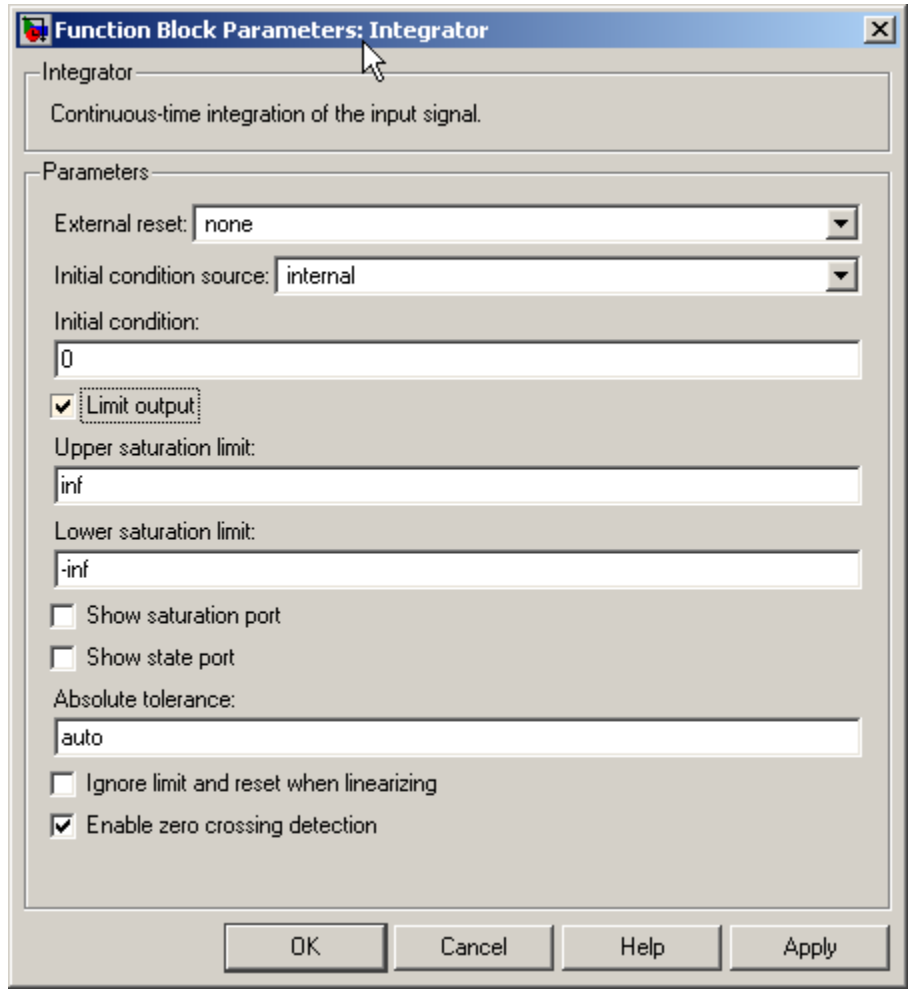
does have downstream impact, consider replacing it using a configurable subsystem when trimming.

Transport Delay Blocks

Transport Delay blocks can cause additional problems when you use the Padé approximation option for linearization within a multi-rate model. In this case, the discretization of the Padé approximation has a frequency response that does not match the frequency response of the original Transport Delay. This can lead to linearized models which do not behave as expected. A solution to this problem is to discretize the transport delay first, using the Model Discretizer, and then linearize the model using the Padé approximation. See "Model Discretizer" in the Simulink documentation for more information on using the Model Discretizer.

Integrator Blocks Near Saturation or a Reset Point

When an Integrator block has an external reset condition or output limitations (saturation) and the model is operating near the point where the Integrator is reset or the output is limited, it might be more meaningful for the linearization to ignore the effect of the saturation or reset. To linearize a model around an operating point that causes the integrator to reset or saturate, select **Ignore limit and reset when linearizing** in the Integrator block parameters dialog box. Selecting this option causes the linearization to treat this block as unresettable and as having no limits on its output, regardless of the settings of the block's reset and output limitation (saturation) options.



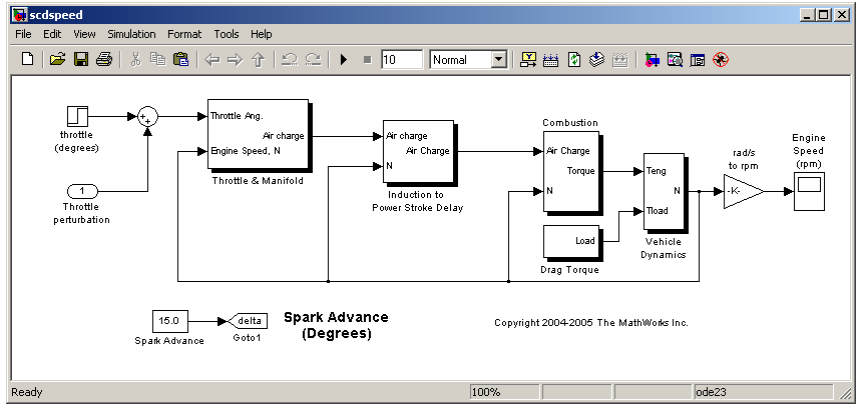
Event-Based Models and Triggered Subsystems

The linearization of triggered subsystems and other event-based models can be particularly difficult because of the system's dependence on previous events. In particular, the execution of a triggered system depends on previous signal events such as zero crossings. Therefore, for linearization, which takes place at a particular moment in time, a trigger event will never happen. Thus, while the event-based dynamics contribute to the definition of the system's

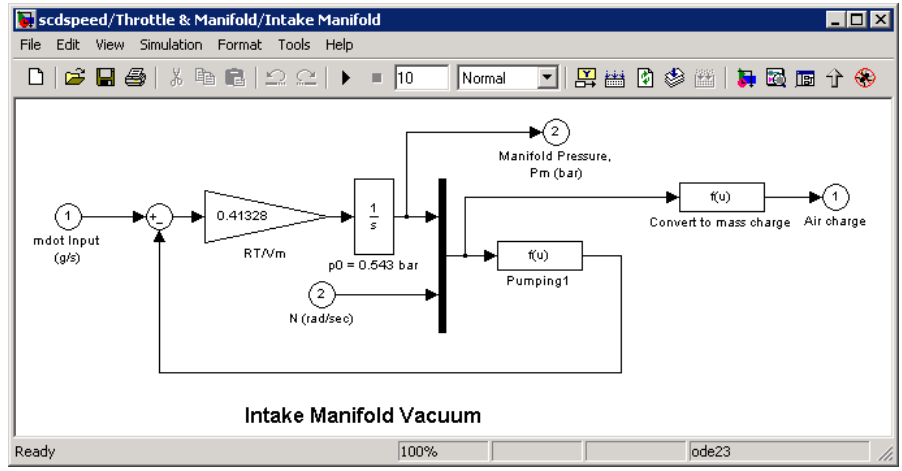
operating point, this information is not captured by the list of values of states and inputs that typically describe the operating point for linearization.

Triggered events describe many different systems. One such system is an internal combustion (IC) engine. When an engine piston approaches the top of a compression stroke, a spark is introduced and combustion occurs. The timing of the spark for combustion is dependent on the speed and position of the engine crankshaft. An example of a Simulink model that models this behavior is `engine.mdl` which is included as a demonstration model in Simulink.

In `engine.mdl`, triggered subsystems generate events when the pistons reach both the top and bottom of the compression stroke. The linearization will not be meaningful because of the presence of these triggered subsystems. However, you can get a meaningful linearization while still preserving the simulation behavior by recasting the event-based dynamics. For example, you can use curve fitting to approximate the event-based behavior. This is done in `scdspeed.mdl`, a demonstration model included in Simulink Control Design and shown in the figure below:



The basic functional approximation in `scdspeed` is included within the Convert to mass charge block inside the subsystem `scdspeed/Throttle & Manifold/Intake Manifold` where a quadratic polynomial is used to approximate the relationship between the Air Charge, the Manifold Pressure, and the Engine Speed.



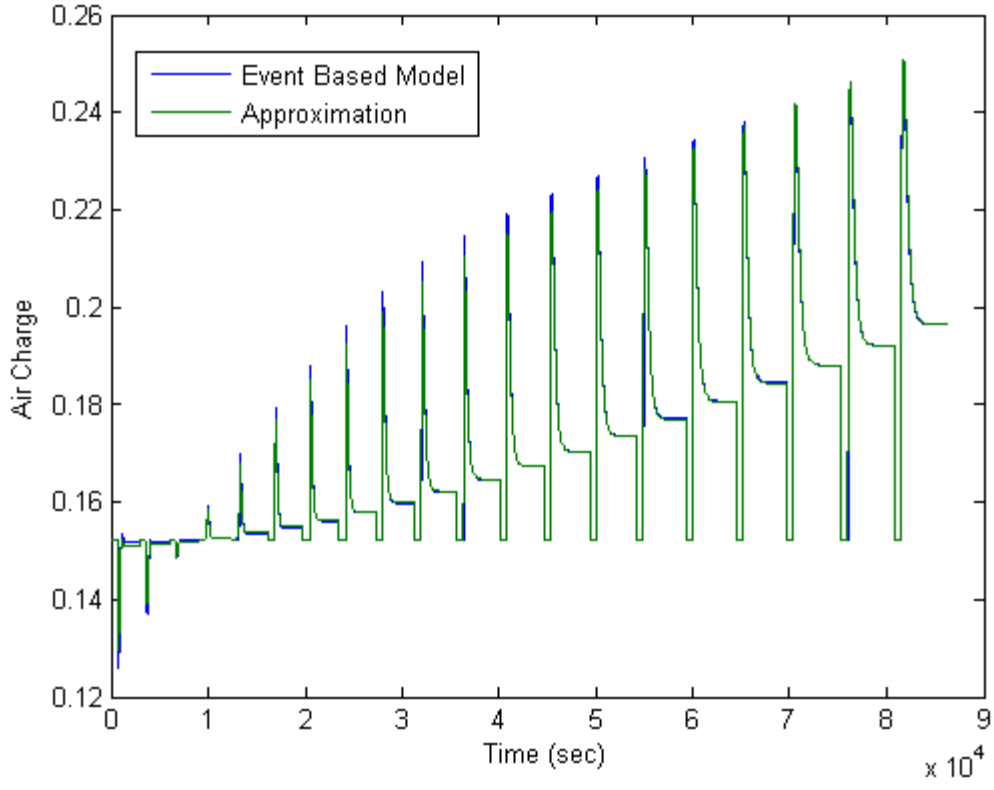
The approximation has the following form:

$$Air\ Charge = p_1 \times Engine\ Speed + p_2 \times Manifold\ Pressure + p_3 \times (Manifold\ Pressure)^2 + p_4 \times Manifold\ Pressure \times Engine\ Speed + p_5$$

Simulation data from the original model is used to compute the unknown parameters p_1, p_2, p_3, p_4, and p_5 using a least squares fitting technique.

When measured data for internal signals is available, you can use Simulink Parameter Estimation to compute the unknown parameters. This method is outlined in the recorded webinar,"New & Upgraded Simulink Tools for Developing More Accurate Models & Better Tuned Control Systems". The webinar also contains a demonstration of the linearization of this model and the use of the linearization to design a feedback controller.

The approximated model can now accurately simulate and linearize the engine from approximately 1500 to 5500 RPM. The following figure shows the comparison between a simulation of the original event-based model, and a simulation of the new approximated model.



Computing Operating Points for SimMechanics Models

When computing operating points (trimming) for a SimMechanics model, you first need to put it in trimming mode. To do this

1. Locate and open the machine environment (Env) block for the system.
2. Within the **Parameters** panel, set **Analysis mode** to `Trimming`. Click **OK** to close the block dialog. This will create an output port in the model that contains constraints related to errors in the system that must be set to zero for a steady state operating point.

3 To set these constraints to zero within a project for the model in the Control and Estimation Tools Manager, select **Operating Points** in the panel on the left, then select the**Compute Operating Points > Outputs** panel. Within this panel, set all constraints to `0`.

At this point you can enter other design specifications on the states and inputs, and then compute an operating point for your model. After you have finished computing operating points for the SimMechanics model, make sure that you reset the **Analysis mode** to `Forward dynamics` in the Env block dialog.

Choosing Initial Values for Computing Operating Points

When you compute an operating point from design specifications (trimming), it is often important to begin with a set of state and input values that are close to the actual steady state operating point values that you are trying to compute. To do this you can simulate the model for a specified period of time and then take a *snapshot* of the state and input values at that time. You can do this using either the Control and Estimation Tools Manager, see "Extracting Operating Points from Simulation" in the online documentation for more information, or using the `findop` command line function, see "Extracting Values from Simulation" in the online documentation for more information.

You can then use the values from the simulation snapshot as initial values for an operating point that you compute from specifications using optimization methods. To initialize the operating point specifications using these snapshot values, click the **Import Initial Values** button in the **Compute Operating Points** panel of the Control and Estimation Tools Manager, or use the `initopspec` function. See "Importing Initial Values" in the online documentation for more information on importing initial values for an operating point in the Control and Estimation Tools Manager.