Simulink® Control Design

For Use with Simulink®

- Modeling
- Simulation
- Implementation

Reference

Version 1

The MathWorks

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

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

Simulink Control Design Reference

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)

Contents

Functions — Categorical List

Linearization Analysis I/Os

get	Get properties of linearization I/Os and operating points
getlinio	Get linearization I/O settings for Simulink® model
linio	Construct linearization I/O settings for Simulink model
set	Set properties of linearization I/Os and operating points
setlinio	Assign I/O settings to Simulink model

Operating Points

addoutputspec	Add output specification to operating point specification
copy	Create copy of operating point or operating point specification
findop	Find operating points from specifications or simulation
get	Get properties of linearization I/Os and operating points
getxu	Extract states and inputs from operating points
initopspec	Initialize operating point specification values
operpoint	Create operating point for Simulink model
operspec	Create operating point specifications for Simulink model

set	Set properties of linearization I/Os and operating points
setxu	Set states and inputs in operating points
update	Update operating point object with structural changes in model

Linearization

findop	Find operating points from specifications or simulation
getlinio	Get linearization I/O settings for Simulink® model
getlinplant	Compute open loop plant model from Simulink diagram
linearize	Create linearized model from Simulink model
linio	Construct linearization I/O settings for Simulink model
linoptions	Set options for linearization and finding operating points
operpoint	Create operating point for Simulink model
operspec	Create operating point specifications for Simulink model

Functions — Alphabetical List

Purpose

Add output specification to operating point specification

Syntax

```
opnew=addoutputspec(op,'block',portnumber)
```

Graphical Interface

As an alternative to the addoutputspec function, add output specifications with the Simulink Control Design GUI. See "Constraining Outputs".

Description

opnew=addoutputspec(op,'block',portnumber) adds an output specification for a Simulink model to an existing operating point specification, op, created with operspec. The signal being constrained by the output specification is indicated by the name of the block, 'block', and the port number, portnumber, that it originates from. You can edit the output specification within the new operating point specification object, opnew, to include the actual constraints or specifications for the signal. Use the new operating point specification object with the function findop to find operating points for the model.

This function will automatically compile the Simulink model, given in the property Model of op, to find the block's output portwidth.

Example

Create an operating point specification for the model magball.

```
op=operspec('magball')
```

This returns the object op. Note that there are no outports in this model and no outputs in the object op.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:            0
      spec:  dx = 0,  initial guess:            0
(2.) magball/Magnetic Ball Plant/Current
```

```
      spec:  dx = 0,  initial guess:            7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:            0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:         0.05

Inputs: None

Outputs: None
```

To add an output specification to the signal between the Controller block and the Magnetic Ball Plant block, use the function `addoutputspec`.

```
newop=addoutputspec(op,'magball/Controller',1)
```

The output specification is added to the operating point specification object.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:            0
      spec:  dx = 0,  initial guess:            0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:            7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:            0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:         0.05

Inputs: None

Outputs:
-----------
```

```
(1.) magball/Controller
      spec:  none
```

Edit the output specification to constrain this signal to be 14.

```
newop.Outputs(1).Known=1, newop.Outputs(1).y=14
```

MATLAB® displays the final output specification.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:             0
      spec:  dx = 0,  initial guess:             0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:             7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:             0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:          0.05

Inputs: None

Outputs:
-----------
(1.) magball/Controller
      spec:  y = 14
```

See Also

findop, operspec, operpoint

Purpose Create copy of operating point or operating point specification

Syntax

```
op_point2=copy(op_point1)
op_spec2=copy(op_spec1)
```

Description

`op_point2=copy(op_point1)` returns a copy of the operating point object `op_point1`. You can create `op_point1` with the function `operpoint`.

`op_spec2=copy(op_spec1)` returns a copy of the operating point specification object `op_spec1`. You can create `op_spec1` with the function `operspec`.

Note The command `op_point2=op_point1` does not create a copy of `op_point1` but creates a pointer to `op_point1`. In this case any changes made to `op_point2` will also be made to `op_point1`.

Example

Create an operating point object for the model, `magball`.

```
opp=operpoint('magball')
```

MATLAB displays the operating point.

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
```

```
      x: 0.05

Inputs: None
```

Create a copy of this object, opp.

```
new_opp=copy(opp)
```

MATLAB displays an exact copy of the object.

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

See Also

operpoint, operspec

Purpose

Find operating points from specifications or simulation

Syntax

```
[op_point,op_report]=findop('model',op_spec)
[op_point,op_report]=findop('model',op_spec,options)
op_point=findop('model',times)
```

Graphical Interface

As an alternative to the findop function, create operating points from specifications or simulation within the **Operating Points** node of the Simulink Control Design GUI. See "Computing Operating Points from Specifications" and "Extracting Operating Points from Simulation".

Remarks

Finding operating points from specifications using the findop function is the same as trimming, or performing trim analysis. Use the findop function instead of the Simulink trim function when working with Simulink Control Design operating point objects and specification objects.

Description

[op_point,op_report]=findop('model',op_spec) finds an operating point, op_point, of the model, 'model', from specifications given in op_spec.

[op_point,op_report]=findop('model',op_spec,options) finds an operating point, op_point, of the model, 'model', from specifications given in op_spec. Several options for the optimization are specified in the options object, which you can create with the function linoptions.

The input to findop, op_spec, is an operating point specification object. Create this object with the function operspec. Specifications on the operating points, such as minimum and maximum values, initial guesses, and known values, are specified by editing op_spec directly or by using get and set. To find equilibrium, or steady-state, operating points, set the SteadyState property of the states and inputs in op_spec to 1. The findop function uses optimization to find operating points that closely meet the specifications in op_spec. By default, findop uses the optimizer graddescent_elim. To use a different optimizer, change the value of OptimizerType in options using the linoptions function.

A report object, `op_report`, gives information on how closely `findop` meets the specifications. The function `findop` displays the report automatically, even if the output is suppressed with a semi-colon. To turn off the display of the report, set `DisplayReport` to `'off'` in `options` using the function `linoptions`.

`op_point=findop('model',times)` runs a simulation of the model, `'model'`, and extracts operating points from the simulation at the *snapshot* times given in the vector, `times`. An operating point object, `op_point`, is returned.

For all syntaxes, the output of `findop` is an operating point object. Use this object with the function `linearize` to create linearized models of Simulink models. The operating point object has the following properties:

- "Model" on page 2-8
- "States" on page 2-8
- "Inputs" on page 2-9
- "Time" on page 2-9

Model

`Model` specifies the name of the Simulink model that this operating point object refers to.

States

`States` describes the operating points of states in the Simulink model. The `States` property is a vector of state objects that contains the operating point values of the states. There is one state object per block that has a state in the Simulink model. The `States` object has the following properties:

`Nx`	Number of states in the block. This property is read-only.
`Block`	Block that the states are associated with

`x`	Vector containing the values of states in the block
`Description`	String describing the block

Inputs

`Inputs` is a vector of input objects that contains the input levels at the operating point. There is one input object per root level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	Inport block that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	String describing the input

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

The operating point report object, returned when finding operating points from specifications, has the following properties:

- `Model`
- `Inputs`
- `Outputs`
- `States`
- `Time`
- `TerminationString`
- `OptimizationOutput`

Of these properties, Model, Inputs, Outputs, States, and Time contain the same information as the operating point specification object, with the addition of dx values for the States and yspec values, or desired y values, for the Outputs. The TerminationString contains the message that findop displays after terminating the optimization. The OptimizationOutput property contains the same properties returned in the output variable of the Optimization Toolbox functions fmincon, fminsearch, and lsqnonlin. See the Optimization Toolbox documentation for more information. If you do not have the Optimization Toolbox, you can access the documentation at

```
http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml
```

Examples

Example 1

Create an operating point specification object for the model magball with the operspec function.

```
op_spec=operspec('magball');
```

Edit the operating point specification object to reflect any specifications on the operating points such as minimum and maximum values, initial guesses, and known values. This example uses the default specifications in which SteadyState is set to 1 for all states, specifying that an equilibrium operating point is desired.

Find the equilibrium operating points with the findop function.

```
op_point=findop('magball',op_spec)
```

This returns an operating point object, op_point.

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

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

```
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

MATLAB displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the operating point values of the states. In this example there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` field although there are no inputs in this model. To view the properties of `op_point` in more detail, use the `get` function.

MATLAB also displays the operating point report object.

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

Operating condition specifications were successfully met.

States:
----------
(1.) magball/Controller/Controller
      x:            0      dx:            0 (0)
      x:    -2.56e-006     dx:            0 (0)
(2.) magball/Magnetic Ball Plant/Current
      x:            7      dx:            0 (0)
(3.) magball/Magnetic Ball Plant/dhdt
      x:            0      dx:   -1.78e-015 (0)
(4.) magball/Magnetic Ball Plant/height
      x:         0.05      dx:            0 (0)
```

```
Inputs: None

Outputs: None
```

In addition to the operating point values, the report shows how closely the specifications were met. In the report above, the `dx` values are all small and close to the desired `dx` values of `0` indicating that an equilibrium or steady-state value was found.

Example 2

To extract an operating point from a simulation at the times 10 and 20, you can use `findop` in the following way.

```
op_point=findop('magball',[10,20])
```

This returns the message

```
There is more than one operating point.  Select an element
in the vector of operating points to display.
```

To display the first operating point, enter the command

```
op_point(1)
```

This should display

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=10)

States:
----------
(1.) magball/Controller/Controller
      x: -4.82e-010
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 2.6e-006
```

```
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

To display the second operating point, enter

```
op_point(2)
```

This returns

```
 Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=20)

States:
----------
(1.) magball/Controller/Controller
      x: -5.5e-010
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 2.54e-006
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

See Also

operspec, linearize

Purpose

Get properties of linearization I/Os and operating points

Syntax

```
get(ob)
get(ob,'PropertyName')
ob.PropertyName
```

Graphical Interface

As an alternative to the `get` function, view properties of linearization I/Os and operating points with the Simulink Control Design GUI. See "Inspecting Analysis I/Os" and "Specifying Operating Points".

Description

`get(ob)` displays all properties and corresponding values of the object, `ob`, which can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`get(ob,'PropertyName')` returns the value of the property, `PropertyName`, within the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`ob.PropertyName` is an alternative notation for displaying the value of the property, `PropertyName`, of the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

Examples

Create an operating point object, `op`, for the Simulink model, `magball`.

```
op=operpoint('magball');
```

Get a list of all object properties using the `get` function with the object name as the only input.

```
get(op)
```

This returns the properties of `op` and their current values.

```
    Model: 'magball'
    States: [4x1 opcond.StatePoint]
    Inputs: []
      Time: 0
```

To view the value of a particular property of op, supply the property name as an argument to get. For example, to view the name of the model associated with the operating point object, type

```
V=get(op,'Model')
```

which returns

```
V =
magball
```

Since op is a structure, you can also view any properties or fields using dot-notation, as in this example.

```
W=op.States
```

This returns a vector of objects containing information about the states in the operating point.

```
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

Use get to view details of W. For example

```
get(W(2),'x')
```

returns

```
ans =
    7.0036
```

See Also

`findop`, `getlinio`, `linio`, `operpoint`, `operspec`, `set`

Purpose

Get linearization I/O settings for Simulink® model

Syntax

```
io = getlinio('sys')
```

Graphical Interface

As an alternative to the `getlinio` function, view linearization I/Os in the **Analysis I/Os** panel of the **Linearization Task** node within the Simulink Control Design GUI. See "Inspecting Analysis I/Os".

Description

`io = getlinio('sys')` finds all linearization annotations in the Simulink model, `sys`, and returns a vector of objects, `io`. Each object represents a linearization annotation in the model and is associated with an output port of a Simulink block. Before running `getlinio`, use the right click menu to insert the linearization annotations, or I/Os, on the signal lines of the model diagram.

Each object within the vector, `io`, has the following properties:

`Active`	`'on'` when the I/O will be used for linearization and `'off'` otherwise
`Block`	Name of the block the I/O is associated with
`OpenLoop`	`'on'` when the feedback loop at the I/O is open and `'off'` when it is closed
`PortNumber`	Integer referring to the output port the I/O is associated with
`Type`	Linearization I/O type • `'in'`: linearization input point • `'out'`: linearization output point • `'outin'`: linearization output then input point • `'inout'`: linearization input then output point
`Description`	String description of the I/O object

You can edit this I/O object to change its properties. Alternatively, you can change the properties of `io` using the `set` function. To upload an edited I/O object to the Simulink model diagram, use the `setlinio` function. Use I/O objects with the function `linearize` to create linear models.

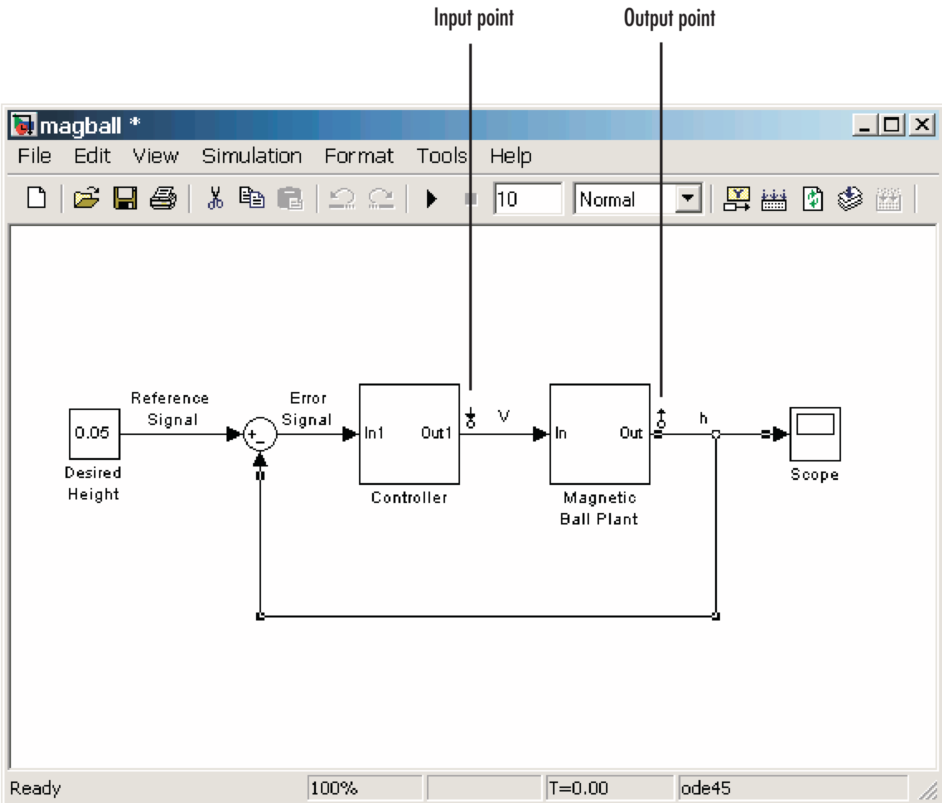
Example

Before creating a vector of I/O objects using `getlinio`, you must add linearization annotations representing the I/Os, such as input points or output points, to a Simulink model.

Open the Simulink model `magball` by typing

```
magball
```

at the MATLAB prompt. Right-click the signal line between the Magnetic Ball Plant and the Controller. Select **Linearization Points -> Input Point** from the menu to place an input point on this signal line. A small arrow pointing towards a small circle just above the signal line represents the input point. Right-click the signal line after the Magnetic Ball Plant. Select **Linearization Points -> Output Point** from the menu to place an output point on this signal line. A small arrow pointing away from a small circle just above the signal line represents the output point. The model diagram should now look like that in the following figure.



To create a vector of I/O objects for this model, type

```
io=getlinio('magball')
```

This returns a formatted display of the linearization I/Os.

```
    Linearization IOs:
--------------------------
Block magball/Controller, Port 1 is marked with the following
properties:
 - No Loop Opening
```

```
  - An Input Perturbation

Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
  - An Output Measurement
  - No Loop Opening
```

There are two entries in the vector, `io`, representing the two linearization annotations previously set in the model diagram. MATLAB displays the name of the block associated with the I/O, the port number associated with the I/O, the type of IO (input perturbation or output measurement referring to an input point or output point respectively), and whether the IO is also a loop opening. By default, the I/Os have no loop openings. Display the properties of each I/O object in more detail using the `get` function.

See Also

`get`, `linearize`, `linio`, `set`, `setlinio`

Purpose

Compute open loop plant model from Simulink diagram

Syntax

```
[sysp,sysc] = getlinplant(block,op)
[sysp,sysc] = getlinplant(block,op,options)
```

Description

`[sysp,sysc] = getlinplant(block,op)` Computes the open loop plant seen by a Simulink block labeled `block` (where `block` specifies the full path to the block). The plant model, `sysp`, and linearized block, `sysc`, are linearized at the operating point `op`.

`[sysp,sysc] = getlinplant(block,op,options)` Computes the open loop plant seen by a Simulink block labeled `block`, using the linearization options specified in `options`.

Example

To compute the open loop model seen by the Controller block in the Simulink model `magball`, first create an operating point object using the function `findop`. In this case the operating point is found from simulation of the model.

```
op=findop('magball',20);
```

Next, compute the open loop model seen by the block `magball/Controller`, with the `getlinplant` function.

```
[sysp,sysc]=getlinplant('magball/Controller',op)
```

The output variable `sysp` gives the open loop plant model as shown below.

```
a =
                   magball/Magn  magball/Magn  magball/Magn
   magball/Magn            -100             0             0
   magball/Magn          -2.798             0         195.7
   magball/Magn               0             1             0

b =
                   magball/Cont
   magball/Magn              50
```

```
   magball/Magn            0
   magball/Magn            0

c =
                 magball/Magn  magball/Magn  magball/Magn
   Controller (             0             0            -1

d =
                 magball/Cont
   Controller (             0

Continuous-time model.
```

See Also

findop, linoptions, operpoint, operspec

Purpose Extract states and inputs from operating points

Syntax

```
x = getxu(op_point)
[x,u] = getxu(op_point)
[x,u,xstruct] = getxu(op_point)
```

Description

`x = getxu(op_point)` extracts a vector of state values, `x`, from the operating point object, `op_point`. The ordering of states in `x` is the same as that used by Simulink.

`[x,u] = getxu(op_point)` extracts a vector of state values, `x`, and a vector of input values, `u`, from the operating point object, `op`. The ordering of states in `x`, and inputs in `u`, is the same as that used by Simulink.

`[x,u,xstruct] = getxu(op_point)` extracts a vector of state values, `x`, a vector of input values, `u`, and a structure of state values, `xstruct`, from the operating point object, `op_point`. The structure of state values, `xstruct`, has the same format as that returned from a Simulink simulation. The ordering of states in `x` and `xstruct`, and inputs in `u`, is the same as that used by Simulink.

Example

Create an operating point object for the `magball` model by typing

```
op=operpoint('magball');
```

To view the states within this operating point, type

```
op.States
```

which returns

```
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
```

```
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

To extract a vector of state values, with the states in the ordering that is compatible with Simulink, along with inputs and a state structure, type

```
[x,u,xstruct]=getxu(op)
```

This returns

```
x =
    0.0500
         0
         0
    7.0036
         0

u =
     []

xstruct =
       time: 0
    signals: [1x4 struct]
```

View `xstruct` in more detail by typing

```
xstruct.signals
```

This displays

```
1x4 struct array with fields:
    values
    dimensions
    label
    blockname
```

View each component of the structure individually. For example:

```
xstruct.signals(1).values

ans =

     0
     0
```

or

```
xstruct.signals(2).values

ans =

    7.0036
```

You can import these vectors and structures into Simulink as initial conditions or input vectors, or use them with `setxu`, to set state and input values in another operating point.

See Also

`operpoint`, `operspec`

Purpose

Initialize operating point specification values

Syntax

```
opnew=initopspec(opspec,oppoint)
opnew=initopspec(opspec,x,u)
opnew=initopspec(opspec,xstruct,u)
```

Graphical Interface

As an alternative to the `initopspec` function, initialize operating point specification values in the **Create Operating Points** panel in the **Operating Points** node within the Simulink Control Design GUI. See "Computing Operating Points from Specifications".

Description

`opnew=initopspec(opspec,oppoint)` initializes the operating point specification object, `opspec`, with the values contained in the operating point object, `oppoint`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. Create `oppoint` with the function `operpoint` or `findop`.

`opnew=initopspec(opspec,x,u)` initializes the operating point specification object, `opspec`, with the values contained in the state vector, `x`, and the input vector, `u`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. You can use the function `getxu` to create `x` and `u` with the correct ordering.

`opnew=initopspec(opspec,xstruct,u)` initializes the operating point specification object, `opspec`, with the values contained in the state structure, `xstruct`, and the input vector, `u`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. You can use the function `getxu` to create `xstruct` and `u` with the correct ordering. Alternatively, `xstruct`, can be saved to the MATLAB workspace after a simulation of the model. See the Simulink documentation for more information on these structures.

Example

Create on operating point using `findop` by simulating the `magball` model and extracting the operating point after 20 time units.

```
oppoint=findop('magball',20)
```

This returns the following operating point.

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=20)

States:
----------
(1.) magball/Controller/Controller
      x: 5.28e-009
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 6.99
(3.) magball/Magnetic Ball Plant/dhdt
      x: -2.62e-005
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

Use these operating point values as initial values in an operating point specification object.

```
opspec=operspec('magball');
newopspec=initopspec(opspec,oppoint)
```

The new operating point specification object is displayed.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:     5.28e-009
      spec:  dx = 0,  initial guess:    -2.56e-006
(1.) magball/Magnetic Ball Plant/Current
```

```
      spec:  dx = 0,  initial guess:          6.99
(1.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:    -2.62e-005
(1.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:          0.05

Inputs: None

Outputs: None
```

You can now use this object to find operating points by optimization.

See Also

`findop`, `getxu`, `operpoint`, `operspec`

Purpose

Create linearized model from Simulink model

Syntax

```
lin=linearize('sys',op,io)
lin=linearize('sys',op,io,options)
lin_block=linearize('sys',op,'blockname')
lin=linearize('sys',op)
lin=linearize('sys',op,options)
[lin,op] = linearize('sys',snapshottimes);
```

Graphical Alternative

As an alternative to the `linearize` function, create linearized models using the **Linearization Task** node of the Simulink Control Design GUI. See "Linearizing the Model".

Description

`lin=linearize('sys',op,io)` takes a model name, `'sys'`, an operating point object, op, and an I/O object, io, as inputs and returns a linear time-invariant state-space model, lin. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. Both `op` and `io` must be associated with the same Simulink model, `sys`.

`lin=linearize('sys',op,io,options)` takes a model name, `'sys'`, an operating point object, op, an I/O object, io, and a linearization options object, `options`, as inputs and returns a linear time-invariant state-space model, `lin`. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. Both `op` and `io` must be associated with the same Simulink model, `sys`. The linearization options object is created with the function `linoptions` and contains several options for linearization.

`lin_block=linearize('sys',op,'blockname')` takes a model name, `'sys'`, an operating point object, op, and the name of a block in the model, `'blockname'`, as inputs and returns `lin_block`, a linear time-invariant state-space model of the named block. The operating point object is created with the function `operpoint` or `findop`. Both `op` and `'blockname'` must be associated with the same Simulink model, `sys`. You can also supply a fourth argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

`lin=linearize('sys',op)` creates a linearized model, `lin`, of the system `'sys'` at the operating point, `op`. Root-level inport and outport blocks in `sys` are used as inputs and outputs for linearization. The operating point object, `op`, is created with the function `operpoint` or `findop`. You can also supply a third argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

`lin=linearize('sys',op,options)` is the form of the `linearize` function that is used with numerical-perturbation linearization. The function returns a linear time-invariant state-space model, `lin`, of the entire model, `sys`. The operating point object, `op`, is created with the function `operpoint` or `findop`. The `LinearizationAlgorithm` option must be set to `'numericalpert'` within `options` for numerical-perturbation linearization to be used. Create the variable `options` with the `linoptions` function. The function uses inport and outport blocks in the model as inputs and outputs for linearization.

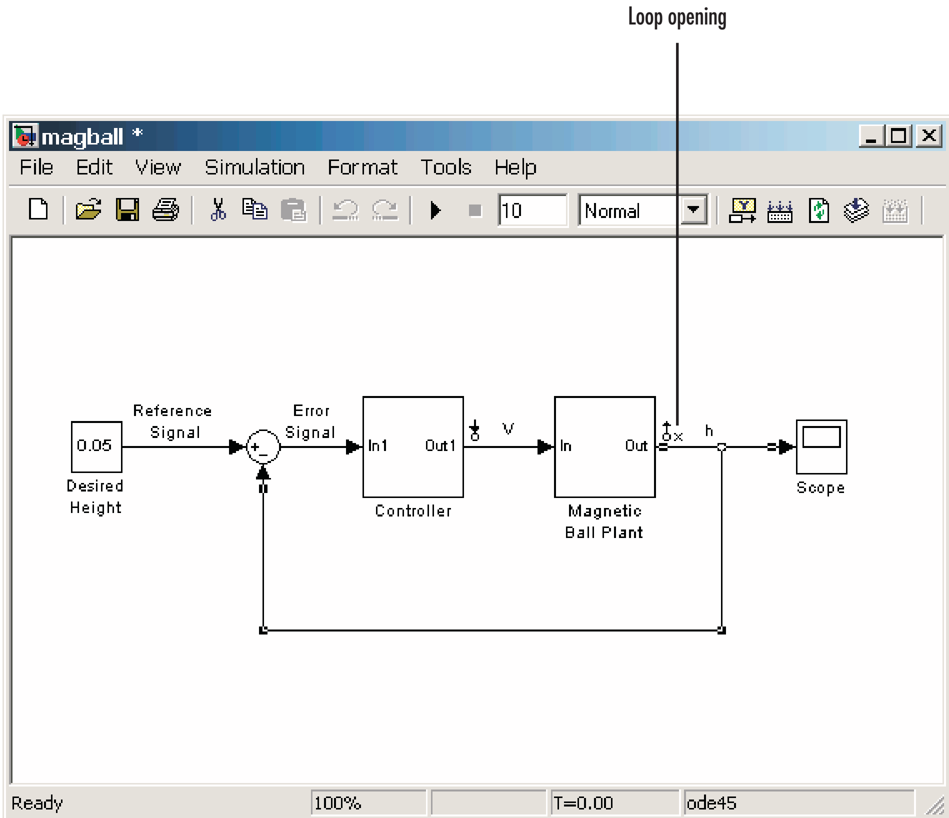
`[lin,op] = linearize('sys',snapshottimes);` creates operating points for the linearization by simulating the model, `'sys'`, and taking snapshots of the system's states and inputs at the times given in the vector `snapshottimes`. The function returns `lin`, a set of linear time-invariant state-space models evaluated and `op`, the set of operating point objects used in the linearization. You can specify input and output points for linearization by providing an additional argument such as a linearization I/O object created with getlinio or linio, or a block name. If an I/O object or block name is not supplied the linearization will use root-level inport and outport blocks in the model. You can also supply an additional argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

Algorithms

Linearization algorithm options are set with the function `linoptions` and passed to the function `linearize` as an optional argument.

Example

Open the Simulink model, `magball`, and insert linearization annotations as shown in the following figure.



Create an I/O object based on the linearization annotations, create an operating point specification object for the model, and then find the operating point using `findop`.

```
io=getlinio('magball');
op=operspec('magball');
op=findop('magball',op);
```

Compute a linear model of the magball system, based on the linearization I/Os, io, and defined about the operating point, op, with the command

```
lin=linearize('magball',op,io)
```

which returns

```
a =
                   magball/Magn  magball/Magn  magball/Magn
   magball/Magn               0             0             1
   magball/Magn               0          -100             0
   magball/Magn           196.2        -2.801             0


b =
                   magball/Cont
   magball/Magn               0
   magball/Magn              50
   magball/Magn               0


c =
                   magball/Magn  magball/Magn  magball/Magn
   magball/Magn               1             0             0


d =
                   magball/Cont
   magball/Magn               0

Continuous-time model.
```

The matrices, a, b, c, and d are the state-space matrices of the linear system given by the following equations

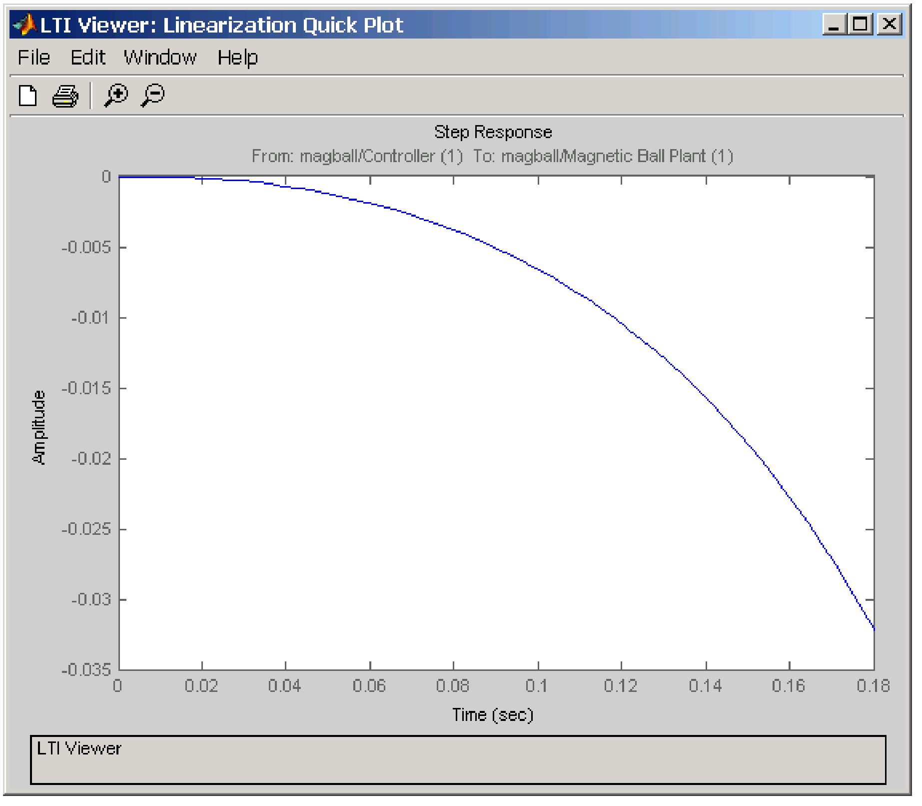
$$\dot{x}(t) = ax(t) + bu(t)$$
$$y(t) = cx(t) + du(t)$$

where $x(t)$ is a vector of states and $u(t)$ is a vector of inputs to the system.

You can view the linearized model, `lin`, with the LTI Viewer

```
ltiview(lin)
```

which produces the following plot.



See Also `findop`, `getlinio`, `operpoint`, `operspec`, `linio`, `linoptions`, `ltiview`

Purpose

Construct linearization I/O settings for Simulink model

Syntax

```
io=linio('blockname',portnum)
io=linio('blockname',portnum,type)
io=linio('blockname',portnum,type,openloop)
```

Graphical Alternative

As an alternative to the `linio` function, create linearization I/O settings by using the right-click menu on the model diagram. See "Inserting Linearization Points".

Description

`io=linio('blockname',portnum)` creates a linearization I/O object for the signal that originates from the outport with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The default I/O type is `'in'`, and the default `OpenLoop` property is `'off'`. Use `io` with the function `linearize` to create linearized models.

`io=linio('blockname',portnum,type)` creates a linearization I/O object for the signal that originates from the outport with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The linearization I/O has the type given by `type`. A list of available types is given below. The default `OpenLoop` property is `'off'`. Use `io` with the function `linearize` to create linearized models.

`io=linio('blockname',portnum,type,openloop)` creates a linearization I/O object for the signal that originates from the outport with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The linearization I/O has the type given by `type` and the open loop status is given by `openloop`. A list of available types is given below. The `openloop` property is set to `'off'` when the I/O is not an open loop point and is set to `'on'` when the I/O is an open loop point. Use `io` with the function `linearize` to create linearized models.

Available linearization I/O types are

- `'in'`, linearization input point
- `'out'`, linearization output point
- `'inout'`, linearization input then output point

- 'outin', linearization output then input point
- 'none', no linearization input/output point

To upload the settings in the I/O object to the Simulink model, use the setlinio function.

Example

Create a linearization I/O setting for the signal line originating from the Controller block of the magball model.

```
io(1)=linio('magball/Controller',1)
```

This displays

```
   Linearization IOs:
--------------------------
Block magball/Controller, Port 1 is marked with the following
properties:
 - No Loop Opening
 - An Input Perturbation
```

By default, this I/O is an input point. Create a second I/O setting within the object, io. This I/O originates from the Magnetic Ball Plant block, is an output point, and is also an open loop point.

```
io(2)=linio('magball/Magnetic Ball Plant',1,'out','on')
```

The new object, io, is displayed.

```
    Linearization IOs:
--------------------------
Block magball/Controller, Port 1 is marked with the following
properties:
 - No Loop Opening
 - An Input Perturbation

Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
```

```
- An Output Measurement
- A Loop Opening
```

See Also

`getlinio`, `linearize`, `setlinio`

linoptions

Purpose

Set options for linearization and finding operating points

Syntax

```
opt=linoptions
opt=linoptions('Property1','Value1','Property2','Value2',...)
```

Graphical Interface

As an alternative to the `linoptions` function, set options for linearization and finding operating points with the Simulink Control Design GUI. See "Changing Linearization Settings" and "Changing Optimization Settings".

Description

`opt=linoptions` creates a linearization options object with the default settings. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

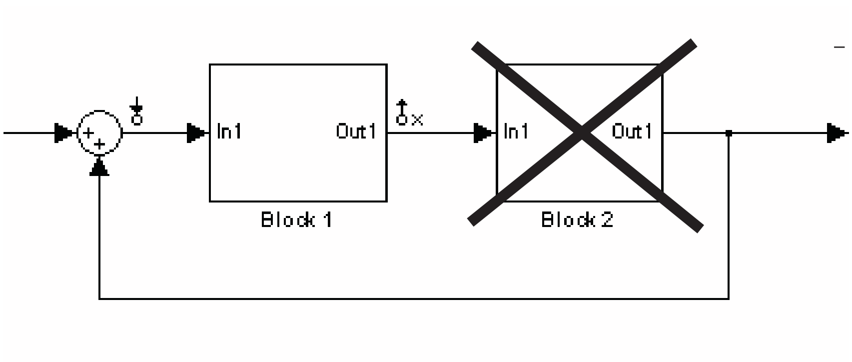
`opt=linoptions('Property1','Value1','Property2','Value2',...)` creates a linearization options object, `opt`, in which the option given by `Property1` is set to the value given in `Value1`, the option given by `Property2` is set to the value given in `Value2`, etc. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

The following options can be set with `linoptions`:

`LinearizationAlgorithm`	Set to `'numericalpert'` (default is `'blockbyblock'`) to enable numerical-perturbation linearization (as in Simulink 3.0) where root level inports and states are numerically perturbed. Linearization annotations are ignored and root level inports and outports are used instead.
`SampleTime`	The time at which the signal is sampled. Nonzero for discrete systems, `0` for continuous systems, `-1` (default) to use the longest sample time that contributes to the linearized model.

`UseFullBlockNameLabels` Set to `'off'` (default) to use truncated names for the linearization I/Os and states in the linearized model. Set to `'on'` to use the full block path to name the linearization I/Os and states in the linearized models.

`BlockReduction` Set to `'on'` (default) to eliminate from the linearized model, blocks that are not in the path of the linearization, as in the following figure. Set to `'off'` to include these blocks in the linearized model.



`IgnoreDiscreteStates` Set to `'on'` to remove any discrete states from the linearization. Set to `'off'` (default) to include discrete states.

`RateConversionMethod` Set to `'zoh'` (default) to use the zero order rate conversion routine when linearizing a multirate system. Set to `'tustin'` to use the Tustin (bilinear) method. Set to `'prewarp'` to use the Tustin approximation with prewarping.

`PreWarpFreq` The critical frequency Wc (in rad/sec) used by the `'prewarp'` option when linearizing a multirate system.

`NumericalPertRel`	Set the perturbation level for obtaining the linear model (default value is `1e-5`). The perturbation of the system's states is specified by $\text{NumericalPertRel} + 1e - 3 \times \text{NumericalPertRel} \times \lvert x \rvert$ The perturbation of the system's inputs is specified by $\text{NumericalPertRel} + 1e - 3 \times \text{NumericalPertRel} \times \lvert u \rvert$
`NumericalXPert`	Individually set the perturbation levels for the system's states.
`NumericalUPert`	Individually set the perturbation levels for the system's inputs.
`OptimizationOptions`	Set options for use with the optimization algorithms. These options are the same as those set with `optimset`. See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox, you can access the documentation at `http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml`
`OptimizerType`	Set optimizer type to be used by trim optimization if the Optimization Toolbox is installed. The available optimizer types are • `graddescent_elim`, the default optimizer, based on the Optimization Toolbox function `fmincon`, enforces an equality constraint to force time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and constraints on output signals. This optimizer fixes states, `x`, and inputs, `u`, by not allowing these variables to be optimized. • `graddescent`, enforces an equality constraint to force time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and constraints on output signals. Minimize the error between the desired (known) values of states, `x`, inputs, `u`, and outputs, `y`. If there are no constraints on `x`, `u`, or `y`, `findop` will attempt to minimize the deviation between the initial guesses for `x` and `u` and the trimmed values.

	• `lsqnonlin` fixes states, `x`, and inputs, `u`, by not allowing these variables to be optimized. The algorithm then tries to minimize the error in `dx/dt` and outputs, `y`. • `simplex` uses the same cost function as `lsqnonlin` with the `fminsearch` optimization routine. See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox, you can access the documentation at `www.mathworks.com/support/`.
`DisplayReport`	Set to `'on'` to display the operating point summary report when running `findop`. Set to `'off'` to suppress the display of this report

See Also

`findop`, `linearize`

Purpose

Create operating point for Simulink model

Syntax

```
op = operpoint('sys')
```

Graphical Interface

As an alternative to the `operpoint` function, create operating points in the **Operating Points** node of the Simulink Control Design GUI. See "Specifying Operating Points".

Description

`op = operpoint('sys')` returns an object, `op`, containing the operating point of a Simulink model, `sys`. Use the object with the function `linearize` to create linearized models. The operating point object properties are

- "Model" on page 2-42
- "States" on page 2-42
- "Inputs" on page 2-43
- "Time" on page 2-43

Edit the properties of this object directly or with the `set` function.

Model

`Model` specifies the name of the Simulink model that this operating point object refers to.

States

`States` describes the operating points of states in the Simulink model. The `States` property is a vector of state objects that contains the operating point values of the states. There is one state object per block that has a state in the Simulink model. The `States` object has the following properties:

`Nx`	Number of states in the block. This property is read-only.
`Block`	Block that the states are associated with

`x`	Vector containing the values of states in the block
`Description`	String describing the block

Inputs

`Inputs` is a vector of input objects that contains the input levels at the operating point. There is one input object per root level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	Inport block that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	String describing the input

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

Example

To create an operating point object for the Simulink model `magball`, type

```
op = operpoint('magball')
```

which returns

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
```

```
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

MATLAB displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the values of the states at the operating point. In this example there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

See Also

`get`, `linearize`, `operspec`, `set`, `update`

Purpose

Create operating point specifications for Simulink model

Syntax

```
op_spec = operspec('sys')
```

Graphical Alternative

As an alternative to the `operspec` function, create operating point specifications in the **Operating Points** node of the Simulink Control Design GUI. See "Computing Operating Points from Specifications".

Description

`op_spec = operspec('sys')` returns an operating point specification object, `op`, for a Simulink model, `sys`. Edit the default operating point specifications directly or use `get` and `set`. Use the operating point specifications object with the function `findop` to find operating points based on the specifications. Use these operating points with the function `linearize` to create linearized models.

The operating point specification object properties are

- “Model” on page 2-45
- “States” on page 2-45
- “Inputs” on page 2-46
- “Time” on page 2-47
- “Outputs” on page 2-47

Use the `set` function to edit the properties of this object before running `findop`.

Model

`Model` is the name of the Simulink model that this operating point specification object is associated with.

States

`States` describes the operating point specifications for states in the Simulink model. The `States` property is a vector of state objects that each contain specifications for particular states. There is one state specification object per block that has a state in the model. The `States` object has the following properties:

`Block`	Block that the states are associated with
`x`	Vector containing values of states in the block. Set the corresponding value of `Known` to `1` when these values are known operating point values. Set the corresponding values of `Known` to `0` when these values are initial guesses for the operating point values. The default value of `x` is the initial condition value for the state.
`Nx`	Number of states in the block. This property is read-only.
`Known`	Vector of values set to `1` for states whose operating points are known exactly and set to `0` for states whose operating points are not known exactly. Set the operating point values themselves in the `x` property.
`SteadyState`	Vector of values set to `1` for states whose operating points should be at equilibrium and set to `0` for states whose operating points are not at equilibrium. The default value of `SteadyState` is `1`.
`Min`	Vector containing the minimum values of the corresponding state's operating point
`Max`	Vector containing the maximum values of the corresponding state's operating point
`Description`	String describing the block

Inputs

`Inputs` is a vector of input specification objects that contains specifications for the input levels at the operating point. There is one input specification object per root level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	The inport block that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing values of inputs. Set the corresponding value of `Known` to `1` when these values are known operating point values. Set the corresponding values of `Known` to `0` when these values are initial guesses for the operating point values.
`Known`	Vector of values set to `1` for inputs whose operating points are known exactly and set to `0` for inputs whose operating points are not known exactly. Set the operating point values themselves in the `u` property.
`Min`	Vector containing the minimum values of the corresponding input's operating point
`Max`	Vector containing the maximum values of the corresponding input's operating point
`Description`	String describing the input

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

Outputs

`Outputs` is a vector of output specification objects that contains the specifications for the output levels at the operating point. There is one output specification object per root level outport block in the Simulink model. To constrain additional outputs, use the `addoutputspec` function to add an another output specification to the operating point specification object. The `Outputs` object has the following properties:

Block	Outport block that the output vector is associated with
PortWidth	Width of the corresponding outport
PortNumber	Port number that the output is associated with
y	Vector containing values of outputs. Set the corresponding value of Known to 1 when these values are known operating point values. Set the corresponding values of Known to 0 when these values are initial guesses for the operating point values.
Known	Vector of values set to 1 for outputs whose operating points are known exactly and set to 0 for outputs whose operating points are not known exactly. Set the operating point values themselves in the y property.
Min	Vector containing the minimum values of the corresponding output's operating point
Max	Vector containing the maximum values of the corresponding output's operating point
Description	String describing the output

Example

To create an operating point specification object for the Simulink model magball, type

```
op_spec = operspec('magball')
```

which returns

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

States:
----------
(1.) magball/Controller/Controller
```

```
      spec:  dx = 0,  initial guess:            0
      spec:  dx = 0,  initial guess:            0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:            7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:            0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:         0.05

Inputs: None

Outputs: None
```

MATLAB displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, default operating point values and initial guesses (based on initial conditions of the states), and steady-state specifications. In this example there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. By default, MATLAB sets the `SteadyState` property to `1` and the upper and lower bounds on the operating points to `Inf` and `-Inf` respectively. MATLAB also displays the `Inputs` and `Outputs` although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

See Also

`addoutputspec`, `findop`, `get`, `operspec`, `linearize`, `set` , `update`

Purpose

Set properties of linearization I/Os and operating points

Syntax

```
set(ob)
set(ob,'PropertyName',val)
ob.PropertyName=val
```

Graphical Interface

As an alternative to the `set` function, set properties of linearization I/Os and operating points in the Simulink Control Design GUI. See "Inspecting Analysis I/Os" and "Specifying Operating Points".

Description

`set(ob)` displays all editable properties of the object, `ob`, which can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`set(ob,'PropertyName',val)` sets the property, `PropertyName`, of the object, `ob`, to the value, `val`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`ob.PropertyName=val` is an alternative notation for assigning the value, `val`, to the property, `PropertyName`, of the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

Examples

Create an operating point object for the Simulink model, `magball`.

```
op_cond=operpoint('magball');
```

Use the `set` function to get a list of all editable properties of this object.

```
set(op_cond)
```

This returns the properties of `op_cond`.

```
ans =
```

```
      Model: {}
     States: {}
     Inputs: {}
       Time: {}
```

To set the value of a particular property of op_cond, provide the property name and the desired value of this property as arguments to set. For example, to change the name of the model associated with the operating point object from 'magball' to 'Magnetic Ball', type

```
set(op_cond,'Model','Magnetic Ball')
```

To view the property value and verify that the change was made type

```
op_cond.Model
```

which returns

```
ans =
Magnetic Ball
```

Since op_cond is a structure, you can set any properties or fields using dot-notation. First produce a list of properties of the second States object within op_cond.

```
set(op_cond.States(2))
ans =
             Nx: {}
          Block: {}
              x: {}
    Description: {}
```

Now, use dot-notation to set the x property to 8.

```
op_cond.States(2).x=8;
```

To view the property and verify that the change was made, type

```
op_cond.States(2)
```

which displays

```
(1.) magball/Magnetic Ball Plant/Current
      x: 8
```

See Also findop, get, linio, operpoint, operspec, setlinio

Purpose

Assign I/O settings to Simulink model

Syntax

```
oldio=setlinio('sys',io)
```

Graphical Interface

As an alternative to the `setlinio` function, edit linearization I/Os in the **Analysis I/Os** panel of the **Linearization Task** node within the Simulink Control Design GUI. See "Inspecting Analysis I/Os".

Description

`oldio=setlinio('sys',io)` assigns the settings in the vector of linearization I/O objects, `io`, to the Simulink model, `sys`, where they are represented by annotations on the signal lines. Use the function `getlinio` or `linio` to create the linearization I/O objects. You can save I/O objects to disk in a MAT-file and use them later to restore linearization settings in a model.

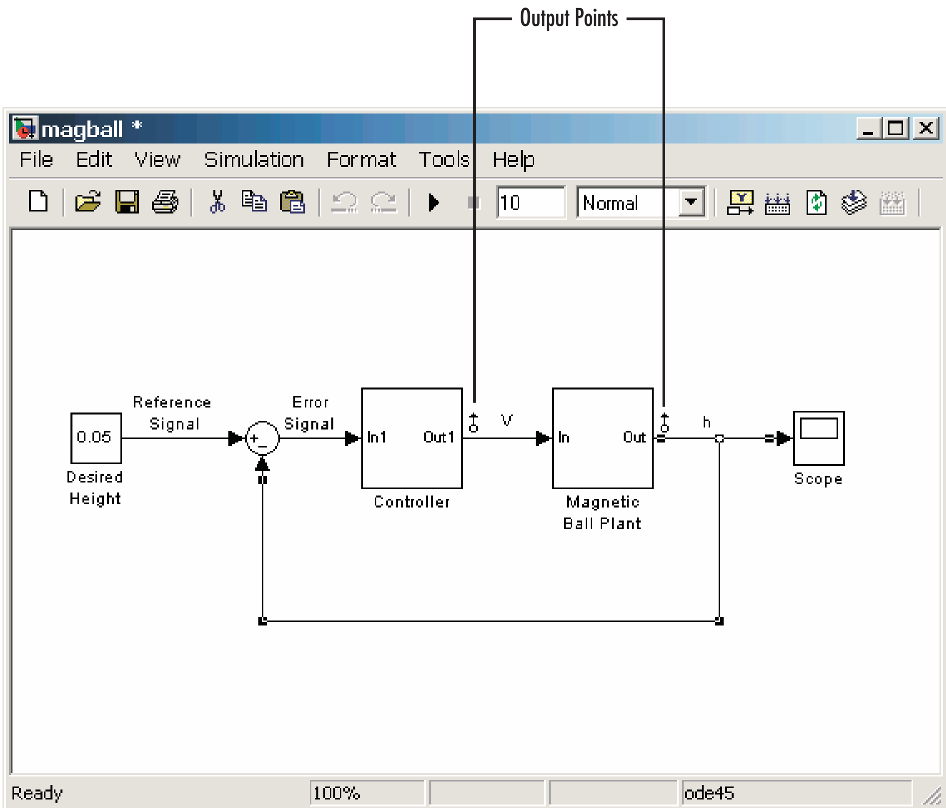
Example

Before assigning I/O settings to a Simulink model using `setlinio`, you must create a vector of I/O objects representing linearization annotations, such as input points or output points, on a Simulink model.

Open the Simulink model `magball` by typing

```
magball
```

at the MATLAB prompt. Right-click the signal line between the Magnetic Ball Plant and the Controller. Select **Linearization Points -> Output Point** from the menu to place an output point on this signal line. A small arrow pointing away from a small circle just above the signal line represents the output point. Right-click the signal line after the Magnetic Ball Plant. Select **Linearization Points -> Output Point** from the menu to place another output point on this signal line. The model diagram should now look like that in the following figure.



Create an I/O object with the `getlinio` function.

```
io=getlinio('magball')
```

Make changes to `io` by editing the object or by using the `set` function. For example:

```
io(1).Type='in';
io(2).OpenLoop='on';
```

Assign the new settings in `io` to the model diagram.

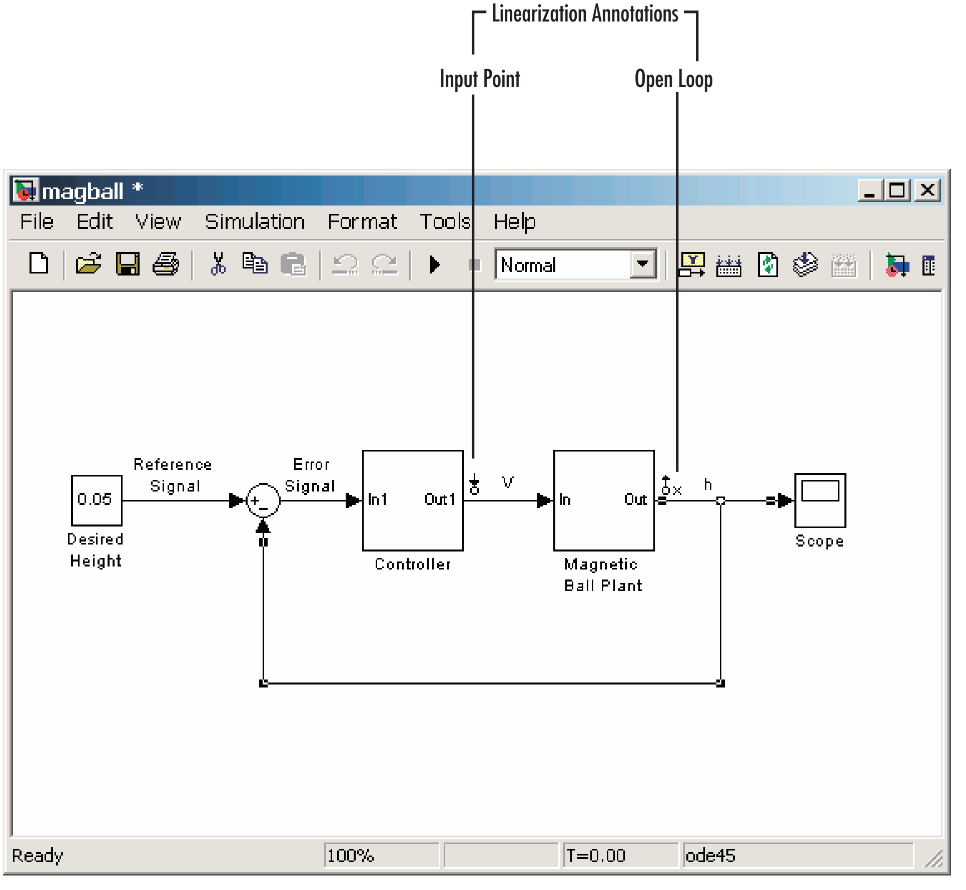
```
oldio=setlinio('magball',io)
```

This returns the old I/O settings (that have been replaced by the settings in io).

```
   Linearization IOs:
--------------------------
Block magball/Controller, Port 1 is marked with the following
properties:
 - An Output Measurement
 - No Loop Opening

Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
 - An Output Measurement
 - No Loop Opening
```

The model diagram should now look like that in the following figure.



See Also

`get`, `getlinio`, `linio`, `set`

Purpose

Set states and inputs in operating points

Syntax

```
op_new=setxu(op_point,x,u)
```

Graphical Alternative

As an alternative to the `setxu` function, set states and inputs of operating points with the Simulink Control Design GUI. See "Importing Operating Points" for more information.

Description

`op_new=setxu(op_point,x,u)` sets the states and inputs in the operating point, `op_point`, with the values in `x` and `u`. A new operating point containing these values, `op_new`, is returned. The variable `x` can be a vector or a structure with the same format as those returned from a Simulink simulation. The variable `u` can be a vector. Both `x` and `u` can be extracted from another operating point object with the `getxu` function.

Example

Open the Simulink model F14 by typing `f14` at the command line. Select **Simulation -> Configuration Parameters -> Data Import/Export**. In the **Save to workspace** panel, select **Final states**. In the **Save options** panel, select `Structure` from **Format**. This will save the final states of the model to the workspace after a simulation.

Start the simulation. After it has run, a new variable, `xFinal`, should be in the workspace. This variable is a structure with two properties, `time` and `signals`.

Create an operating point object for F14 by typing

```
op_point=operpoint('f14')
```

Note that all states are initially set to `0`. Set the states in this object to be the values in `xFinal`. Set the input to be `9`.

```
newop=setxu(op_point,xFinal,9)
```

The new operating point is displayed

```
Operating Point for the Model f14.
 (Time-Varying Components Evaluated at time t=0)
```

```
States:
----------
(1.) f14/Actuator Model
      x: -0.032
(2.) f14/Aircraft Dynamics Model/Transfer Fcn.1
      x: 0.56
(3.) f14/Aircraft Dynamics Model/Transfer Fcn.2
      x: 678
(4.) f14/Controller/Alpha-sensor Low-pass Filter
      x: 0.392
(5.) f14/Controller/Pitch Rate Lead Filter
      x: 0.133
(6.) f14/Controller/Proportional plus integral compensator
      x: 0.166
(7.) f14/Controller/Stick Prefilter
      x: 0.1
(8.) f14/Dryden Wind Gust Models/Q-gust model
      x: 0.114
(9.) f14/Dryden Wind Gust Models/W-gust model
      x: 0.46
      x: -2.05

Inputs:
-----------
(1.) f14/u
      u: 9
```

See Also

getxu, initopspec, operpoint, operspec

Purpose

Update operating point object with structural changes in model

Syntax

```
update(op)
```

Graphical Alternative

As an alternative to the `update` function, update operating point objects with the **Sync with Model** button in the Simulink Control Design GUI. See "Specifying Operating Points" for more information.

Description

`update(op)` updates an operating point object, `op`, to reflect any changes in the associated Simulink model, such as states being added or removed.

Example

Open the magball model

```
magball
```

Create an operating point object for the model.

```
op=operpoint('magball')
```

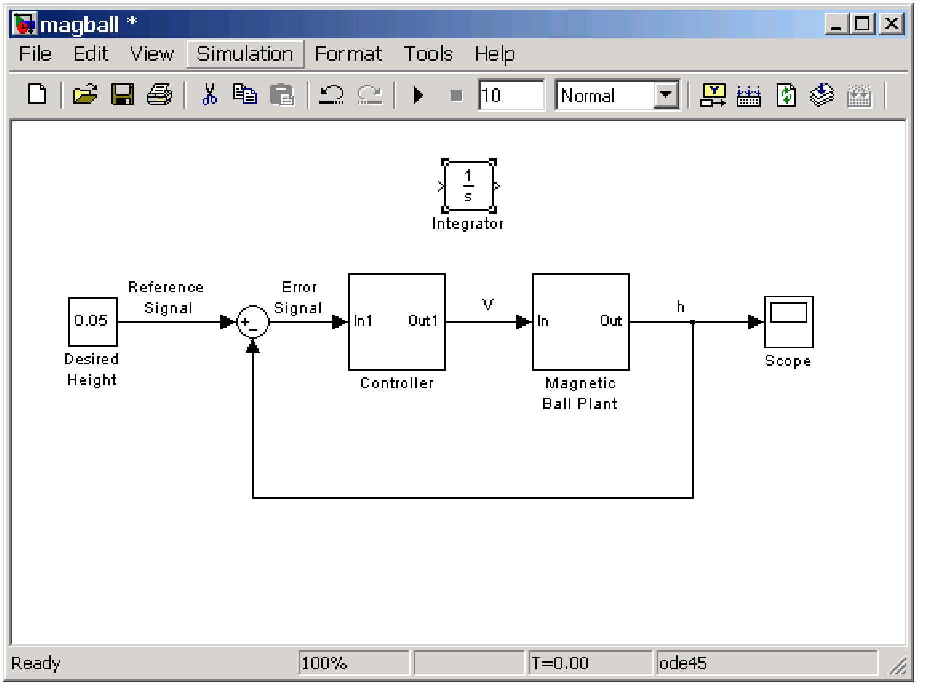
This returns

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

```
Inputs: None
```

Add an `Integrator` block to the model, as shown in the following figure.



Update the operating point to include this new state.

```
update(op)
```

The new operating point is shown below.

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

States:
----------
```

```
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
(5.) magball/Integrator
      x: 0

Inputs: None
```

See Also

operpoint, operspec

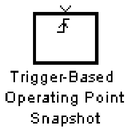
Blocks — Alphabetical List

Trigger-Based Operating Point Snapshot

Purpose Generate operating points and/or linearizations at triggered events

Library Simulink Control Design

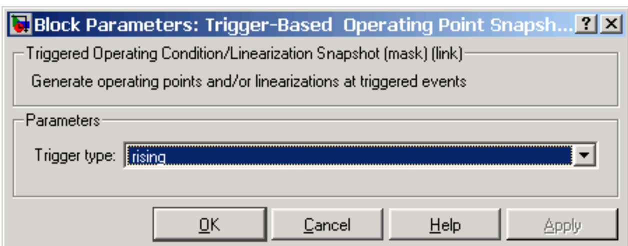
Description



Attach this block to a signal in a model when you want to take a snapshot of the system's operating point at triggered events such as when the signal crosses zero or when the signal sends a function call. You can also perform a linearization at these events. To extract the operating point or perform the linearization you need to simulate the model using either the `findop` or `linearize` functions, or the `simulation snapshots` option in the Control and Estimation Tools Manager.

Choose the trigger type in the **Block Parameters** dialog box, as shown below. The possible trigger types are

- `rising`: the signal crosses zero while increasing
- `falling`: the signal crosses zero while decreasing
- `either`: the signal crosses zero while either increasing or decreasing
- `function-call`: the signal send a function call



See Also `findop`, `linearize`

Index