Model Predictive Control Toolbox™

Getting Started Guide

Alberto Bemporad
N. Lawrence Ricker
Manfred Morari

MATLAB®

R2019**b**

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Revision History

October 2004	First printing	New for Version 2.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 2.2.2 (Release 2006a)
September 2006	Online only	Revised for Version 2.2.3 (Release 2006b)
March 2007	Online only	Revised for Version 2.2.4 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 2.3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.1.1 (Release 2009b)
March 2010	Online only	Revised for Version 3.2 (Release 2010a)
September 2010	Online only	Revised for Version 3.2.1 (Release 2010b)
April 2011	Online only	Revised for Version 3.3 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)
March 2012	Online only	Revised for Version 4.1 (Release 2012a)
September 2012	Online only	Revised for Version 4.1.1 (Release 2012b)
March 2013	Online only	Revised for Version 4.1.2 (Release 2013a)
September 2013	Online only	Revised for Version 4.1.3 (Release 2013b)
March 2014	Online only	Revised for Version 4.2 (Release R2014a)
October 2014	Online only	Revised for Version 5.0 (Release 2014b)
March 2015	Online only	Revised for Version 5.0.1 (Release 2015a)
September 2015	Online only	Revised for Version 5.1 (Release 2015b)
March 2016	Online only	Revised for Version 5.2 (Release 2016a)
September 2016	Online only	Revised for Version 5.2.1 (Release 2016b)
March 2017	Online only	Revised for Version 5.2.2 (Release 2017a)
September 2017	Online only	Revised for Version 6.0 (Release 2017b)
March 2018	Online only	Revised for Version 6.1 (Release 2018a)
September 2018	Online only	Revised for Version 6.2 (Release 2018b)
March 2019	Online only	Revised for Version 6.3 (Release 2019a)
September 2019	Online only	Revised for Version 6.3.1 (Release 2019b)

Contents

3

Design MPC Controllers

Introduction

- "Model Predictive Control Toolbox Product Description" on page 1-2
- "Acknowledgments" on page 1-3
- "Bibliography" on page 1-4

Model Predictive Control Toolbox Product Description

Design and simulate model predictive controllers

Model Predictive Control Toolbox provides functions, an app, and Simulink® blocks for designing and simulating model predictive controllers (MPCs). The toolbox lets you specify plant and disturbance models, horizons, constraints, and weights. By running closed-loop simulations, you can evaluate controller performance.

You can adjust the behavior of the controller by varying its weights and constraints at run time. To control a nonlinear plant, you can implement adaptive and gain-scheduled MPCs. For applications with fast sample rates, you can generate an explicit model predictive controller from a regular controller or implement an approximate solution.

For rapid prototyping and embedded system implementation, the toolbox supports automatic C-code and IEC 61131-3 Structured Text generation.

Key Features

- App for interactive design of MPC controllers
- Runtime adjustment of weights and constraints
- Adaptive and gain-scheduled MPCs for controlling systems with nonlinear dynamics
- Explicit MPC and approximate solution for guaranteed worst-case execution time
- Economic MPC with arbitrary nonlinear cost function and constraints
- Computationally efficient quadratic programming (QP) solver, and support for third-party solvers
- Support for C-code generation (with Simulink Coder™) and IEC 61131-3 Structured Text generation (with Simulink PLC Coder™)

Acknowledgments

MathWorks would like to acknowledge the following contributors to Model Predictive Control Toolbox.

Alberto Bemporad

Professor of Control Systems, IMT Institute for Advanced Studies Lucca, Italy. Research interests include model predictive control, hybrid systems, optimization algorithms, and applications to automotive, aerospace, and energy systems. Fellow of the IEEE®. Author of the Model Predictive Control Simulink library and commands.

N. Lawrence Ricker

Professor of Chemical Engineering, University of Washington, Seattle, USA. Research interests include model predictive control and process optimization. Author of the Model Predictive Control Simulink library and commands.

Manfred Morari

Professor at the Automatic Control Laboratory and former Head of Department of Information Technology and Electrical Engineering, ETH Zurich, Switzerland. Research interests include model predictive control, hybrid systems, and robust control. Fellow of the IEEE, AIChE, and IFAC. Co-author of the first version of the toolbox.

Bibliography
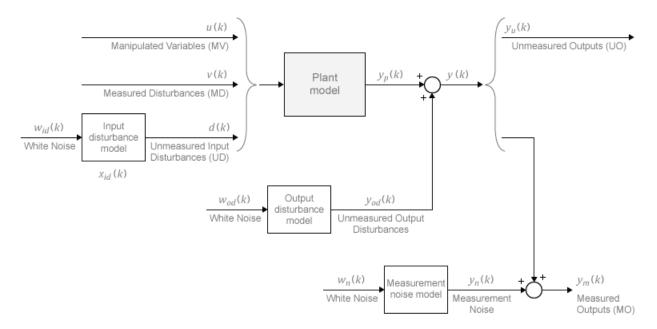
[1] Allgower, F., and A. Zheng, *Nonlinear Model Predictive Control*, Springer-Verlag, 2000.

[2] Camacho, E. F., and C. Bordons, *Model Predictive Control*, Springer-Verlag, 1999.

[3] Kouvaritakis, B., and M. Cannon, *Non-Linear Predictive Control: Theory & Practice*, IEE Publishing, 2001.

[4] Maciejowski, J. M., *Predictive Control with Constraints*, Pearson Education POD, 2002.

[5] Prett, D., and C. Garcia, *Fundamental Process Control*, Butterworths, 1988.

[6] Rossiter, J. A., *Model-Based Predictive Control: A Practical Approach*, CRC Press, 2003.

2

Building Models

MPC Modeling

Model predictive controllers use plant, disturbance, and noise models for prediction and state estimation. The model structure used in an MPC controller appears in the following illustration.

Plant Model

You can specify the plant model in one of the following linear-time-invariant (LTI) formats:

- Numeric LTI models — Transfer function (`tf`), state space (`ss`), zero-pole-gain (`zpk`)
- Identified models (requires System Identification Toolbox™) — `idss`, `idtf`, `idproc`, and `idpoly`

The MPC controller performs all estimation and optimization calculations using a discrete-time, delay-free, state-space system with dimensionless input and output variables. Therefore, when you specify a plant model in the MPC controller, the software performs the following, if needed:

1 Conversion to state space — The `ss` command converts the supplied model to an LTI state-space model.

2 Discretization or resampling — If the model sample time differs from the MPC controller sample time (defined in the `Ts` property), one of the following occurs:

- If the model is continuous time, the `c2d` command converts it to a discrete-time LTI object using the controller sample time.

- If the model is discrete time, the `d2d` command resamples it to generate a discrete-time LTI object using the controller sample time.

3 Delay removal — If the discrete-time model includes any input, output, or internal delays, the `absorbDelay` command replaces them with the appropriate number of poles at $z = 0$, increasing the total number of discrete states. The `InputDelay`, `OutputDelay`, and `InternalDelay` properties of the resulting state-space model are all zero.

4 Conversion to dimensionless input and output variables — The MPC controller enables you to specify a scale factor for each plant input and output variable. If you do not specify scale factors, they default to `1`. The software converts the plant input and output variables to dimensionless form as follows:

$$x_p(k + 1) = A_p x_p(k) + B S_i u_p(k)$$

$$y_p(k) = S_o^{-1} C x_p(k) + S_o^{-1} D S_i u_p(k).$$

where A_p, B, C, and D are the constant zero-delay state-space matrices from step 3, and:

- S_i is a diagonal matrix of input scale factors in engineering units.

- S_o is a diagonal matrix of output scale factors in engineering units.

- x_p is the state vector from step 3 in engineering units (including any absorbed delay states). No scaling is performed on state variables.

- u_p is a vector of dimensionless plant input variables, including manipulated variables, measured disturbances, and unmeasured input disturbances.

- y_p is a vector of dimensionless plant output variables.

The resulting plant model has the following equivalent form:

$$x_p(k + 1) = A_p x_p(k) + B_{pu} u(k) + B_{pv} v(k) + B_{pd} d(k)$$

$$y_p(k) = C_p x_p(k) + D_{pu} u(k) + D_{pv} v(k) + D_{pd} d(k).$$

Here, $C_p = S_o^{-1}C$, B_{pu}, B_{pv}, and B_{pd} are the corresponding columns of BS_i. Also, D_{pu}, D_{pv}, and D_{pd} are the corresponding columns of $S_o^{-1}DS_i$. Finally, $u(k)$, $v(k)$, and $d(k)$ are the dimensionless manipulated variables, measured disturbances, and unmeasured input disturbances, respectively.

The MPC controller enforces the restriction of $D_{pu} = 0$, which means that the controller does not allow direct feedthrough from any manipulated variable to any plant output.

Input Disturbance Model

If your plant model includes unmeasured input disturbances, $d(k)$, the input disturbance model specifies the signal type and characteristics of $d(k)$. See "Controller State Estimation" for more information about the model.

The `getindist` command provides access to the model in use.

The input disturbance model is a key factor that influences the following controller performance attributes:

- Dynamic response to apparent disturbances — The character of the controller response when the measured plant output deviates from its predicted trajectory, due to an unknown disturbance or modeling error.

- Asymptotic rejection of sustained disturbances — If the disturbance model predicts a sustained disturbance, controller adjustments continue until the plant output returns to its desired trajectory, emulating a classical integral feedback controller.

You can provide the input disturbance model as an LTI state-space (`ss`), transfer function (`tf`), or zero-pole-gain (`zpk`) object using `setindist`. The MPC controller converts the input disturbance model to a discrete-time, delay-free, LTI state-space system using the same steps used to convert the plant model on page 2-2. The result is:

$$x_{id}(k + 1) = A_{id}x_{id}(k) + B_{id}w_{id}(k)$$
$$d(k) = C_{id}x_{id}(k) + D_{id}w_{id}(k).$$

where A_{id}, B_{id}, C_{id}, and D_{id} are constant state-space matrices, and:

- $x_{id}(k)$ is a vector of $n_{xid} \geq 0$ input disturbance model states.

- $d_k(k)$ is a vector of n_d dimensionless unmeasured input disturbances.
- $w_{id}(k)$ is a vector of $n_{id} \geq 1$ dimensionless white noise inputs, assumed to have zero mean and unit variance.

If you do not provide an input disturbance model, then the controller uses a default model, which has integrators with dimensionless unity gain added to its outputs. An integrator is added for each unmeasured input disturbance, unless doing so would cause a violation of state observability. In this case, a static system with dimensionless unity gain is used instead.

Output Disturbance Model

The output disturbance model is a special case of the more general input disturbance model. Its output, $y_{od}(k)$, is directly added to the plant output rather than affecting the plant states. The output disturbance model specifies the signal type and characteristics of $y_{od}(k)$, and it is often used in practice. See "Controller State Estimation" for more details about the model.

The `getoutdist` command provides access to the output disturbance model in use.

You can specify a custom output disturbance model as an LTI state-space (`ss`), transfer function (`tf`), or zero-pole-gain (`zpk`) object using `setoutdist`. Using the same steps as for the plant model on page 2-2, the MPC controller converts the specified output disturbance model to a discrete-time, delay-free, LTI state-space system. The result is:

$$x_{od}(k + 1) = A_{od}x_{od}(k) + B_{od}w_{od}(k)$$
$$y_{od}(k) = C_{od}x_{od}(k) + D_{od}w_{od}(k).$$

where A_{od}, B_{od}, C_{od}, and D_{od} are constant state-space matrices, and:

- $x_{od}(k)$ is a vector of $n_{xod} \geq 1$ output disturbance model states.
- $y_{od}(k)$ is a vector of n_y dimensionless output disturbances to be added to the dimensionless plant outputs.
- $w_{od}(k)$ is a vector of n_{od} dimensionless white noise inputs, assumed to have zero mean and unit variance.

If you do not specify an output disturbance model, then the controller uses a default model, which has integrators with dimensionless unity gain added to some or all of its outputs. These integrators are added according to the following rules:

- No disturbances are estimated, that is no integrators are added, for unmeasured plant outputs.
- An integrator is added for each measured output in order of decreasing output weight.

  - For time-varying weights, the sum of the absolute values over time is considered for each output channel.
  - For equal output weights, the order within the output vector is followed.

- For each measured output, an integrator is not added if doing so would cause a violation of state observability. Instead, a gain with a value of zero is used instead.

If there is an input disturbance model, then the controller adds any default integrators to that model before constructing the default output disturbance model.

Measurement Noise Model

One controller design objective is to distinguish disturbances, which require a response, from measurement noise, which should be ignored. The measurement noise model specifies the expected noise type and characteristics. See "Controller State Estimation" for more details about the model.

Using the same steps as for the plant model on page 2-2, the MPC controller converts the measurement noise model to a discrete-time, delay-free, LTI state-space system. The result is:

$$x_n(k + 1) = A_n x_n(k) + B_n w_n(k)$$
$$y_n(k) = C_n x_n(k) + D_n w_n(k).$$

Here, A_n, B_n, C_n, and D_n are constant state space matrices, and:

- $x_n(k)$ is a vector of $n_{xn} \geq 0$ noise model states.
- $y_n(k)$ is a vector of n_{ym} dimensionless noise signals to be added to the dimensionless measured plant outputs.
- $w_n(k)$ is a vector of $n_n \geq 1$ dimensionless white noise inputs, assumed to have zero mean and unit variance.

If you do not supply a noise model, the default is a unity static gain: $n_{xn} = 0$, D_n is an n_{ym}-by-n_{ym} identity matrix, and A_n, B_n, and C_n are empty.

For an `mpc` controller object, `MPCobj`, the property `MPCobj.Model.Noise` provides access to the measurement noise model.

Note If the minimum eigenvalue of $D_n D_n^T$ is less than 1×10^{-8}, the MPC controller adds 1×10^{-4} to each diagonal element of D_n. This adjustment makes a successful default Kalman gain calculation more likely.

See Also

More About

- "Controller State Estimation"
- "Adjust Disturbance and Noise Models"

Signal Types

Inputs

The *plant inputs* are the independent variables affecting the plant. As shown in "MPC Modeling" on page 2-2, there are three types:

Measured disturbances

The controller can't adjust them, but uses them for feedforward compensation.

Manipulated variables

The controller adjusts these in order to achieve its goals.

Unmeasured disturbances

These are independent inputs of which the controller has no direct knowledge, and for which it must compensate.

Outputs

The *plant outputs* are the dependent variables (outcomes) you wish to control or monitor. As shown in "MPC Modeling" on page 2-2, there are two types:

Measured outputs

The controller uses these to estimate unmeasured quantities and as feedback on the success of its adjustments.

Unmeasured outputs

The controller estimates these based on available measurements and the plant model. The controller can also hold unmeasured outputs at setpoints or within constraint boundaries.

You must specify the input and output types when designing the controller. See "Input and Output Types" on page 2-14 for more details.

See Also

More About
- "MPC Modeling" on page 2-2

Construct Linear Time Invariant Models

Model Predictive Control Toolbox software supports the same LTI model formats as does Control System Toolbox software. You can use whichever is most convenient for your application and convert from one format to another. For more details, see "Basic Models" (Control System Toolbox).

Transfer Function Models

A transfer function (TF) relates a particular input/output pair. For example, if $u(t)$ is a plant input and $y(t)$ is an output, the transfer function relating them might be:

$$\frac{Y(s)}{U(s)} = G(s) = \frac{s + 2}{s^2 + s + 10}e^{-1.5s}$$

This TF consists of a *numerator* polynomial, $s+2$, a *denominator* polynomial, s^2+s+10, and a delay, which is 1.5 time units here. You can define G using Control System Toolbox `tf` function:

```
Gtf1 = tf([1 2], [1 1 10],'OutputDelay',1.5)

Transfer function:
                s + 2
exp(-1.5*s) * ------------
             s^2 + s + 10
```

Zero/Pole/Gain Models

Like the TF format, the zero/pole/gain (ZPK) format relates an input/output pair. The difference is that the ZPK numerator and denominator polynomials are factored, as in

$$G(s) = 2.5\frac{s + 0.45}{(s + 0.3)(s + 0.1 + 0.7i)(s + 0.1 - 0.7i)}$$

(zeros and/or poles are complex numbers in general).

You define the ZPK model by specifying the zero(s), pole(s), and gain as in

```
poles = [-0.3, -0.1+0.7*i, -0.1-0.7*i];
Gzpk1 = zpk(-0.45,poles,2.5);
```
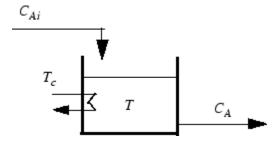
State-Space Models

The state-space format is convenient if your model is a set of LTI differential and algebraic equations. For example, consider the following linearized model of a continuous stirred-tank reactor (CSTR) involving an exothermic (heat-generating) reaction [1].

$$\frac{dC'_A}{dt} = a_{11}C'_A + a_{12}T' + b_{11}T'_c + b_{12}C'_{Ai}$$

$$\frac{dT'}{dt} = a_{21}C'_A + a_{22}T' + b_{21}T'_c + b_{22}C'_{Ai}$$

where C_A is the concentration of a key reactant, T is the temperature in the reactor, T_c is the coolant temperature, C_{Ai} is the reactant concentration in the reactor feed, and a_{ij} and b_{ij} are constants. See the process schematic in "CSTR Schematic" on page 2-11. The primes (e.g., C'_A) denote a deviation from the nominal steady-state condition at which the model has been linearized.

CSTR Schematic

Measurement of reactant concentrations is often difficult, if not impossible. Let us assume that T is a measured output, C_A is an unmeasured output, T_c is a manipulated variable, and C_{Ai} is an unmeasured disturbance.

The model fits the general state-space format

$$\frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

where

$$x = \begin{bmatrix} C'_A \\ T' \end{bmatrix}, \ u = \begin{bmatrix} T'_c \\ C'_{Ai} \end{bmatrix}, \ y = \begin{bmatrix} T' \\ C'_A \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \ B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \ C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \ D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The following code shows how to define such a model for some specific values of the a_{ij} and b_{ij} constants:

```
A = [-0.0285   -0.0014
     -0.0371   -0.1476];
B = [-0.0850    0.0238
      0.0802    0.4462];
C = [0 1
     1 0];
D = zeros(2,2);
CSTR = ss(A,B,C,D);
```

This defines a *continuous-time* state-space model. If you do not specify a sampling period, a default sampling value of zero applies. You can also specify discrete-time state-space models. You can specify delays in both continuous-time and discrete-time models.

Note In the CSTR example, the *D* matrix is zero and the output does not instantly respond to change in the input. The Model Predictive Control Toolbox software prohibits direct (instantaneous) feedthrough from a manipulated variable to an output. For example, the CSTR model could include direct feedthrough from the unmeasured disturbance, C_{Ai}, to either C_A or T but direct feedthrough from T_c to either output would violate this restriction. If the model had direct feedthrough from T_c, you can add a small delay at this input to circumvent the problem.

LTI Object Properties

The `ss` function in the last line of the above code creates a state-space model, CSTR, which is an *LTI object*. The `tf` and `zpk` commands described in "Transfer Function Models" on page 2-10 and "Zero/Pole/Gain Models" on page 2-10 also create LTI objects. Such objects contain the model parameters as well as optional properties.

LTI Properties for the CSTR Example

The following code sets some of the CSTR model's optional properties:

```
CSTR.InputName = {'T_c','C_A_i'};
CSTR.OutputName = {'T','C_A'};
CSTR.StateName = {'C_A','T'};
CSTR.InputGroup.MV = 1;
CSTR.InputGroup.UD = 2;
CSTR.OutputGroup.MO = 1;
CSTR.OutputGroup.UO = 2;
CSTR
```

The first three lines specify labels for the input, output and state variables. The next four specify the signal type for each input and output. The designations MV, UD, MO, and UO mean *manipulated variable*, *unmeasured disturbance*, *measured output*, and *unmeasured output*. (See "Signal Types" on page 2-8 for definitions.) For example, the code specifies that input 2 of model CSTR is an unmeasured disturbance. The last line causes the LTI object to be displayed, generating the following lines in the MATLAB® Command Window:

```
A =
            C_A        T
   C_A  -0.0285  -0.0014
   T    -0.0371  -0.1476

B =
           T_c     C_Ai
   C_A  -0.085   0.0238
   T     0.0802  0.4462

C =
         C_A      T
   T       0      1
   C_A     1      0


D =
         T_c  C_Ai
   T       0     0
   C_A     0     0

Input groups:
    Name     Channels
     MV          1
     UD          2

Output groups:
    Name     Channels
```

```
   MO          1
   UO          2
```

Continuous-time model

Input and Output Names

The optional `InputName` and `OutputName` properties affect the model displays, as in the above example. The software also uses the `InputName` and `OutputName` properties to label plots and tables. In that context, the underscore character causes the next character to be displayed as a subscript.

Input and Output Types

General Case

As mentioned in "Signal Types" on page 2-8, Model Predictive Control Toolbox software supports three input types and two output types. In a Model Predictive Control Toolbox design, designation of the input and output types determines the controller dimensions and has other important consequences.

For example, suppose your plant structure were as follows:

Plant Inputs	Plant Outputs
Two manipulated variables (MVs)	Three measured outputs (MOs)
One measured disturbance (MD)	Two unmeasured outputs (UOs)
Two unmeasured disturbances (UDs)	

The resulting controller has four inputs (the three MOs and the MD) and two outputs (the MVs). It includes feedforward compensation for the measured disturbance, and assumes that you wanted to include the unmeasured disturbances and outputs as part of the regulator design.

If you didn't want a particular signal to be treated as one of the above types, you could do one of the following:

- Eliminate the signal before using the model in controller design.
- For an output, designate it as unmeasured, then set its weight to zero.
- For an input, designate it as an unmeasured disturbance, then define a custom state estimator that ignores the input.

Note By default, the software assumes that unspecified plant inputs are manipulated variables, and unspecified outputs are measured. Thus, if you didn't specify signal types in the above example, the controller would have four inputs (assuming all plant outputs were measured) and five outputs (assuming all plant inputs were manipulated variables).

For model CSTR, the default Model Predictive Control Toolbox assumptions are incorrect. You must set its InputGroup and OutputGroup properties, as illustrated in the above code, or modify the default settings when you load the model into **MPC Designer**.

Use setmpcsignals to make type definition. For example:

```
CSTR = setmpcsignals(CSTR,'UD',2,'UO',2);
```

sets InputGroup and OutputGroup to the same values as in the previous example. The CSTR display would then include the following lines:

```
Input groups:
       Name          Channels
    Unmeasured          2
    Manipulated         1


Output groups:
       Name          Channels
    Unmeasured          2
     Measured           1
```

Notice that setmpcsignals sets unspecified inputs to Manipulated and unspecified outputs to Measured.

LTI Model Characteristics

Control System Toolbox software provides functions for analyzing LTI models. Some of the more commonly used are listed below. Type the example code at the MATLAB prompt to see how they work for the CSTR example.

Example	Intended Result
dcgain(CSTR)	Calculate gain matrix for the CSTR model's input/output pairs.

Example	Intended Result
`impulse(CSTR)`	Graph CSTR model's unit-impulse response.
`linearSystemAnalyzer(CSTR)`	Open the Linear System Analyzer with the CSTR model loaded. You can then display model characteristics by making menu selections.
`pole(CSTR)`	Calculate CSTR model's poles (to check stability, etc.).
`step(CSTR)`	Graph CSTR model's unit-step response.
`zero(CSTR)`	Compute CSTR model's transmission zeros.

References

[1] Seborg, D. E., T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*, 2nd Edition, Wiley, 2004, pp. 34–36 and 94–95.

See Also

setmpcsignals | ss | tf | zpk

More About

- "Specify Multi-Input Multi-Output Plants" on page 2-17

Specify Multi-Input Multi-Output Plants

Most MPC applications involve plants with multiple inputs and outputs. You can use `ss`, `tf`, and `zpk` to represent a MIMO plant model. For example, consider the following model of a distillation column [1], which has been used in many advanced control studies:

$$
\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \dfrac{12.8e^{-s}}{16.7s + 1} & \dfrac{-18.9e^{-3s}}{21.0s + 1} & \dfrac{3.8e^{-8.1s}}{14.9s + 1} \\ \dfrac{6.6e^{-7s}}{10.9s + 1} & \dfrac{-19.4e^{-3s}}{14.4s + 1} & \dfrac{4.9e^{-3.4s}}{13.2s + 1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}
$$

Outputs y_1 and y_2 represent measured product purities. The controller manipulates the inputs, u_1 and u_2, to hold each output at a specified setpoint. These inputs represent the flow rates of reflux and reboiler steam, respectively. Input u_3 is a measured feed flow rate disturbance.

The model consists of six transfer functions, one for each input/output pair. Each transfer function is the first-order-plus-delay form often used by process control engineers.

Specify the individual transfer functions for each input/output pair. For example, `g12` is the transfer function from input u_1 to output y_2.

```
g11 = tf( 12.8, [16.7 1], 'IOdelay', 1.0,'TimeUnit','minutes');
g12 = tf(-18.9, [21.0 1], 'IOdelay', 3.0,'TimeUnit','minutes');
g13 = tf(  3.8, [14.9 1], 'IOdelay', 8.1,'TimeUnit','minutes');
g21 = tf(  6.6, [10.9 1], 'IOdelay', 7.0,'TimeUnit','minutes');
g22 = tf(-19.4, [14.4 1], 'IOdelay', 3.0,'TimeUnit','minutes');
g23 = tf(  4.9, [13.2 1], 'IOdelay', 3.4,'TimeUnit','minutes');
```

Define a MIMO system by creating a matrix of transfer function models.

```
DC = [g11 g12 g13
      g21 g22 g23];
```

Define the input and output signal names and specify the third input as a measured input disturbance.

```
DC.InputName = {'Reflux Rate','Steam Rate','Feed Rate'};
DC.OutputName = {'Distillate Purity','Bottoms Purity'};
DC = setmpcsignals(DC,'MD',3);
```

```
-->Assuming unspecified input signals are manipulated variables.
```

Review the resulting system.

```
DC
```

```
DC =

  From input "Reflux Rate" to output...
                                      12.8
    Distillate Purity:  exp(-1*s) * ----------
                                     16.7 s + 1

                                     6.6
    Bottoms Purity:  exp(-7*s) * ----------
                                 10.9 s + 1

  From input "Steam Rate" to output...
                                      -18.9
    Distillate Purity:  exp(-3*s) * --------
                                     21 s + 1

                                     -19.4
    Bottoms Purity:  exp(-3*s) * ----------
                                 14.4 s + 1

  From input "Feed Rate" to output...
                                        3.8
    Distillate Purity:  exp(-8.1*s) * ----------
                                      14.9 s + 1

                                      4.9
    Bottoms Purity:  exp(-3.4*s) * ----------
                                   13.2 s + 1

Input groups:
      Name         Channels
     Measured         3
    Manipulated     1,2

Output groups:
      Name       Channels
     Measured     1,2

Continuous-time transfer function.
```

References

[1] Wood, R. K., and M. W. Berry, *Chem. Eng. Sci.,* Vol. 28, pp. 1707, 1973.

See Also
setmpcsignals | ss | tf | zpk

Related Examples

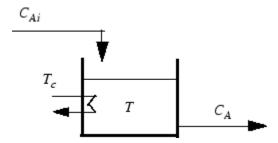- "Construct Linear Time Invariant Models" on page 2-10

CSTR Model

The linearized model of a continuous stirred-tank reactor (CSTR) involving an exothermic (heat-generating) reaction is represented by the following differential equations:

$$\frac{dC'_A}{dt} = a_{11}C'_A + a_{12}T' + b_{11}T'_c + b_{12}C'_{Ai}$$

$$\frac{dT'}{dt} = a_{21}C'_A + a_{22}T' + b_{21}T'_c + b_{22}C'_{Ai}$$

where C_A is the concentration of a key reactant, T is the temperature in the reactor, T_c is the coolant temperature, C_{Ai} is the reactant concentration in the reactor feed, and a_{ij} and b_{ij} are constants. The primes (e.g., C'_A) denote a deviation from the nominal steady-state condition at which the model has been linearized.

Measurement of reactant concentrations is often difficult, if not impossible. Let us assume that T is a measured output, C_A is an unmeasured output, T_c is a manipulated variable, and C_{Ai} is an unmeasured disturbance.

The model fits the general state-space format

$$\frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

where

$$x = \begin{bmatrix} C'_A \\ T' \end{bmatrix}, u = \begin{bmatrix} T'_c \\ C'_{Ai} \end{bmatrix}, y = \begin{bmatrix} T' \\ C'_A \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The following code shows how to define such a model for some specific values of the a_{ij} and b_{ij} constants:

```
A = [-0.0285  -0.0014
     -0.0371  -0.1476];
B = [-0.0850   0.0238
      0.0802   0.4462];
C = [0 1
     1 0];
D = zeros(2,2);
CSTR = ss(A,B,C,D);
```

The following code sets some of the CSTR model's optional properties:

```
CSTR.InputName = {'T_c', 'C_A_i'};
CSTR.OutputName = {'T', 'C_A'};
CSTR.StateName = {'C_A', 'T'};
CSTR.InputGroup.MV = 1;
CSTR.InputGroup.UD = 2;
CSTR.OutputGroup.MO = 1;
CSTR.OutputGroup.UO = 2;
```

To view the properties of CSTR, enter:

```
CSTR
```

Linearize Simulink Models

Generally, real systems are nonlinear. To design an MPC controller for a nonlinear system, you can model the plant in Simulink.

Although an MPC controller can regulate a nonlinear plant, the model used within the controller must be linear. In other words, the controller employs a linear approximation of the nonlinear plant. The accuracy of this approximation significantly affects controller performance.

To obtain such a linear approximation, you *linearize* the nonlinear plant at a specified *operating point*.

Note Simulink Control Design software must be installed to linearize nonlinear Simulink models.

You can linearize a Simulink model:

- From the command line.
- Using the Linear Analysis Tool.
- Using **MPC Designer**. For an example, see "Linearize Simulink Models Using MPC Designer" on page 2-32.

Linearization Using MATLAB Code

This example shows how to obtain a linear model of a plant using a MATLAB script.

For this example the CSTR model, `CSTR_OpenLoop`, is linearized. The model inputs are the coolant temperature (manipulated variable of the MPC controller), limiting reactant concentration in the feed stream, and feed temperature. The model states are the temperature and concentration of the limiting reactant in the product stream. Both states are measured and used for feedback control.

Obtain Steady-State Operating Point

The operating point defines the nominal conditions at which you linearize a model. It is usually a steady-state condition.

Suppose that you plan to operate the CSTR with the output concentration, C_A, at 2 $kmol/m^3$. The nominal feed concentration is 10 $kmol/m^3$, and the nominal feed temperature is 300 K. Create an operating point specification object to define the steady-state conditions.

```
opspec = operspec('CSTR_OpenLoop');
opspec = addoutputspec(opspec,'CSTR_OpenLoop/CSTR',2);
opspec.Outputs(1).Known = true;
opspec.Outputs(1).y = 2;

op1 = findop('CSTR_OpenLoop',opspec);

 Operating point search report:
---------------------------------

 Operating point search report for the Model CSTR_OpenLoop.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) CSTR_OpenLoop/CSTR/C_A
      x:                2       dx:      -4.6e-12 (0)
(2.) CSTR_OpenLoop/CSTR/T_K
      x:                373     dx:      5.49e-11 (0)

Inputs:
----------
(1.) CSTR_OpenLoop/Coolant Temperature
      u:                299     [-Inf Inf]

Outputs:
----------
(1.) CSTR_OpenLoop/CSTR
      y:                2     (2)
```

The calculated operating point is C_A = 2 $kmol/m^3$ and T_K = 373 K. Notice that the steady-state coolant temperature is also given as 299 K, which is the nominal value of the manipulated variable of the MPC controller.

To specify:

• Values of known inputs, use the Input.Known and Input.u fields of opspec

- Initial guesses for state values, use the `State.x` field of `opspec`

For example, the following code specifies the coolant temperature as 305 K and initial guess values of the `C_A` and `T_K` states before calculating the steady-state operating point:

```
opspec = operspec('CSTR_OpenLoop');
opspec.States(1).x = 1;
opspec.States(2).x = 400;
opspec.Inputs(1).Known = true;
opspec.Inputs(1).u = 305;

op2 = findop('CSTR_OpenLoop',opspec);

 Operating point search report:
---------------------------------

 Operating point search report for the Model CSTR_OpenLoop.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) CSTR_OpenLoop/CSTR/C_A
      x:             1.78      dx:      -1.42e-14 (0)
(2.) CSTR_OpenLoop/CSTR/T_K
      x:             377       dx:       5.68e-14 (0)

Inputs:
----------
(1.) CSTR_OpenLoop/Coolant Temperature
      u:             305

Outputs: None
----------
```

Specify Linearization Inputs and Outputs

If the linearization input and output signals are already defined in the model, as in `CSTR_OpenLoop`, then use the following to obtain the signal set.

```
io = getlinio('CSTR_OpenLoop');
```

Otherwise, specify the input and output signals as shown here.

```
io(1) = linio('CSTR_OpenLoop/Coolant Temperature',1,'input');
io(2) = linio('CSTR_OpenLoop/Feed Concentration',1,'input');
io(3) = linio('CSTR_OpenLoop/Feed Temperature',1,'input');
io(4) = linio('CSTR_OpenLoop/CSTR',1,'output');
io(5) = linio('CSTR_OpenLoop/CSTR',2,'output');
```

Linearize Model

Linearize the model using the specified operating point, `op1`, and input/output signals, `io`.

```
sys = linearize('CSTR_OpenLoop',op1,io)

sys =

  A =
            C_A      T_K
   C_A       -5  -0.3427
   T_K    47.68    2.785

  B =
        Coolant Temp  Feed Concent  Feed Tempera
   C_A             0             1             0
   T_K           0.3             0             1

  C =
           C_A  T_K
   CSTR/1    0    1
   CSTR/2    1    0

  D =
           Coolant Temp  Feed Concent  Feed Tempera
   CSTR/1             0             0             0
   CSTR/2             0             0             0

Continuous-time state-space model.
```

Linearization Using Linear Analysis Tool in Simulink Control Design

This example shows how to linearize a Simulink model using the Linear Analysis Tool, provided by the Simulink Control Design software.

Open Simulink Model

This example uses the CSTR model, CSTR_OpenLoop.

```
open_system('CSTR_OpenLoop')
```

Specify Linearization Inputs and Outputs

The linearization inputs and outputs are already specified for CSTR_OpenLoop. The input signals correspond to the outputs from the Feed Concentration, Feed Temperature, and Coolant Temperature blocks. The output signals are the inputs to the CSTR Temperature and Residual Concentration blocks.
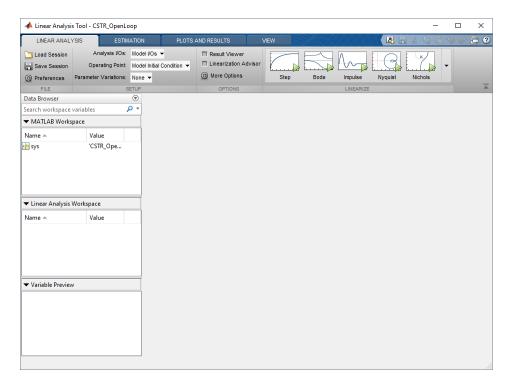
To specify a signal as a linearization input or output, first open the **Linearization** tab. To do so, in the Simulink **Apps** gallery, click **Linearization Manager**. Then, in the Simulink model window, click the signal.

To specify the signal as a:

- Linearization input, on the **Linearization** tab, in the **Insert Analysis Points** gallery, click **Input Perturbation**.
- Linearization output, on the **Linearization** tab, in the **Insert Analysis Points** gallery, click **Output Measurement**.
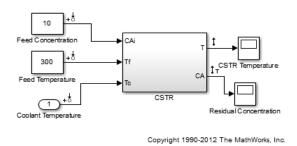
Open Linear Analysis Tool

To open the **Linear Analysis Tool**, in the **Apps** gallery, click **Model Linearizer**.

Specify Residual Concentration as Known Trim Constraint
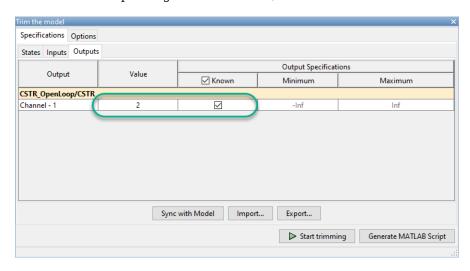
In the Simulink model window, click the CA output signal from the CSTR block. Then, on the **Linearization** tab, in the **Insert Analysis Points** gallery, click **Trim Output Constraint**.

Copyright 1990-2012 The MathWorks, Inc.

In the **Linear Analysis Tool**, on the **Linear Analysis** tab, select **Operating Point > Trim Model**.

In the Trim the model dialog box, on the **Outputs** tab:
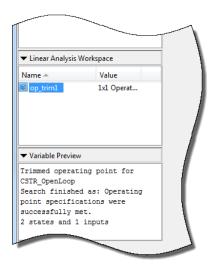
* Select the **Known** check box for Channel - 1 under **CSTR_OpenLoop/CSTR**.
* Set the corresponding **Value** to 2 kmol/m^3.

Create and Verify Operating Point
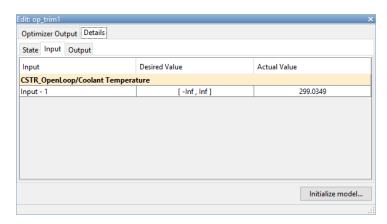
In the Trim the model dialog box, click **Start trimming**.

The operating point op_trim1 displays in the **Linear Analysis Workspace**.

Double click `op_trim1` to view the resulting operating point.
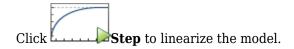
In the Edit dialog box, select the **Input** tab.
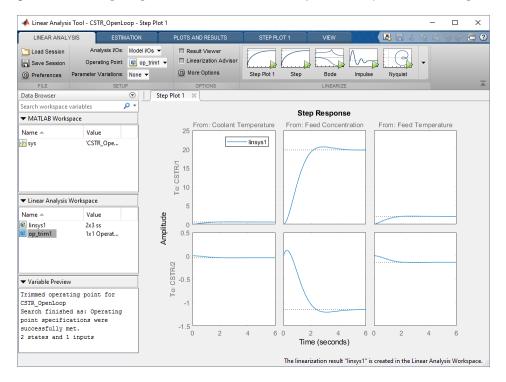
The coolant temperature at steady state is 299 K, as desired.

Linearize Model

On the **Linear Analysis** tab, in the **Operating Point** drop-down list, select `op_trim1`.

Click **Step** to linearize the model.

This option creates the linear model `linsys1` in the **Linear Analysis Workspace** and generates a step response for this model. `linsys1` uses `optrim1` as its operating point.

The step response from feed concentration to output `CSTR/2` displays an interesting inverse response. An examination of the linear model shows that `CSTR/2` is the residual CSTR concentration, `C_A`. When the feed concentration increases, `C_A` increases initially because more reactant is entering, which increases the reaction rate. This rate increase results in a higher reactor temperature (output `CSTR/1`), which further increases the reaction rate and `C_A` decreases dramatically.

Export Linearization Result

If necessary, you can repeat any of these steps to improve your model performance. Once you are satisfied with your linearization result, in the Linear Analysis Tool, drag and drop

it from the **Linear Analysis Workspace** to the **MATLAB Workspace**. You can now use your linear model to design an MPC controller.

See Also
Linear Analysis Tool | `linearize`

Related Examples
- "Design MPC Controller in Simulink" on page 3-38
- "Design Controller Using MPC Designer" on page 3-2
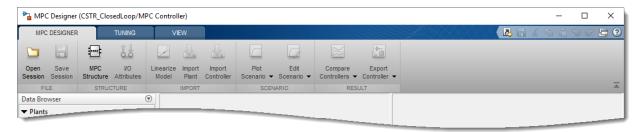- "Design MPC Controller at the Command Line" on page 3-24

Linearize Simulink Models Using MPC Designer

This topic shows how to linearize Simulink models using **MPC Designer**. To do so, open the app from a Simulink model that contains an MPC Controller block. For this example, use the CSTR_ClosedLoop model.

```
sys = 'CSTR_ClosedLoop';
open_system(sys)
```

In the model window, double-click the MPC Controller block.

In the Block Parameters dialog box, ensure that the **MPC Controller** field is empty, and click **Design** to open **MPC Designer**.
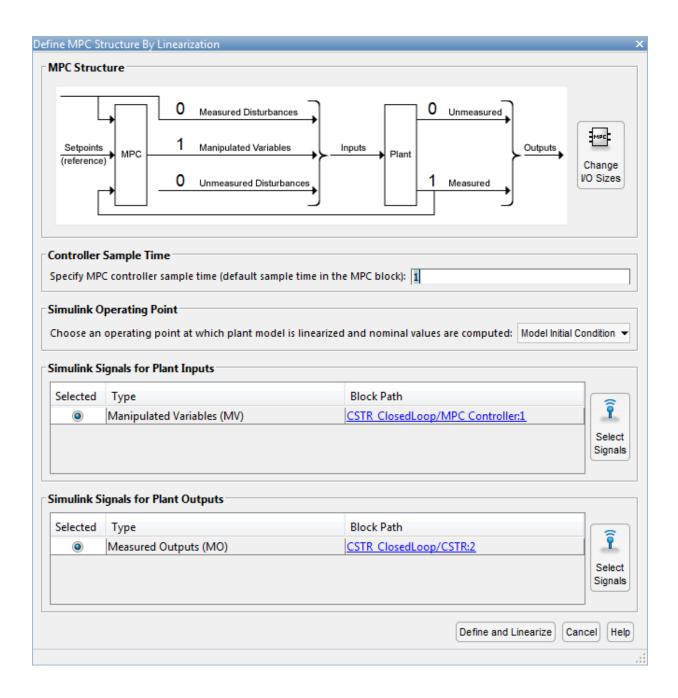
Using **MPC Designer**, you can define the MPC structure by linearizing the Simulink model. After you define the initial MPC structure, you can also linearize the model at different operating points and import the linearized plants.

Note If a controller from the MATLAB workspace is specified in the **MPC Controller** field, the app imports the specified controller. In this case, the MPC structure is derived from the imported controller. In this case, you can still linearize the Simulink model and import the linearized plants.

Define MPC Structure By Linearization
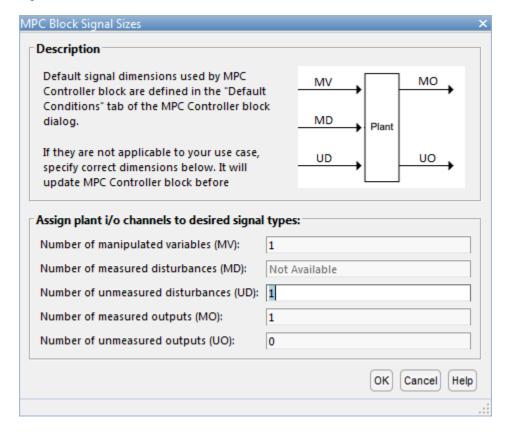
This example shows how to define the plant input/output structure in **MPC Designer** by linearizing a Simulink model.

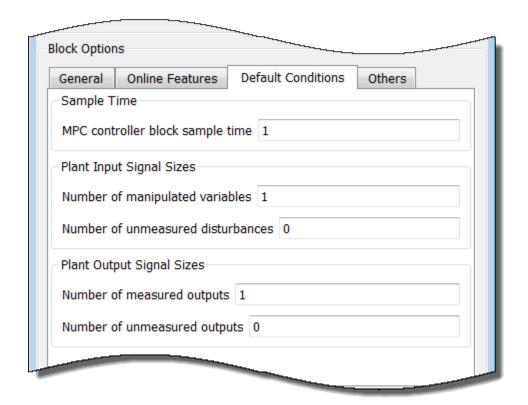On the **MPC Designer** tab, in the **Structure** section, click **MPC Structure**.

Define MPC Structure By Linearization   ✕

MPC Structure

	0 Measured Disturbances		**0** Unmeasured	
Setpoints (reference) → MPC	**1** Manipulated Variables	Inputs → Plant	**0** Unmeasured	Outputs
	0 Unmeasured Disturbances		**1** Measured	

Change I/O Sizes

Controller Sample Time

Specify MPC controller sample time (default sample time in the MPC block): [1]

Simulink Operating Point

Choose an operating point at which plant model is linearized and nominal values are computed: [Model Initial Condition ▼]

Simulink Signals for Plant Inputs

Selected	Type	Block Path
◉	Manipulated Variables (MV)	CSTR_ClosedLoop/MPC Controller:1

Select Signals

Simulink Signals for Plant Outputs

Selected	Type	Block Path
◉	Measured Outputs (MO)	CSTR_ClosedLoop/CSTR:2

Select Signals

[Define and Linearize] [Cancel] [Help]

Specify Signal Dimensions

In the Define MPC Structure By Linearization dialog box, in the **MPC Structure** section, if the displayed signal dimensions do not match your model, click **Change I/O Sizes** to configure the dimensions. Any unmeasured disturbances or unmeasured outputs in your model are not detected by the MPC Controller block. Specify the dimensions for these signals.

Tip In the MPC Controller Block Parameters dialog box, in the **Default Conditions** tab, you can define the controller sample time and signal dimensions before opening **MPC Designer**.

Block Options

| General | Online Features | Default Conditions | Others |

Sample Time

MPC controller block sample time `1`

Plant Input Signal Sizes

Number of manipulated variables `1`

Number of unmeasured disturbances `0`

Plant Output Signal Sizes

Number of measured outputs `1`

Number of unmeasured outputs `0`
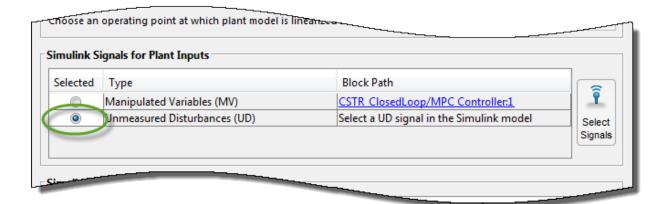
Select Plant Input/Output Signals

Before linearizing the model, assign Simulink signal lines to each MPC signal type in your model. The app uses these signals as linearization inputs and outputs.

In the **Simulink Signals for Plant Inputs** and **Simulink Signals for Plant Outputs** sections, the **Block Path** is automatically defined for manipulated variables, measured outputs, and measured disturbances. **MPC Designer** detects these signals since they are connected to the MPC Controller block. If your application has unmeasured disturbances or unmeasured outputs, select their corresponding Simulink signal lines.
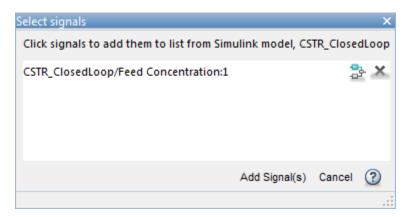
To choose a signal type, use the **Selected** option buttons.

Click **Select Signals**.

In the Simulink model window, click the signal line corresponding to the selected signal type.

The signal is highlighted, and its block path is added to the Select signals dialog box.

In the Select signals dialog box, click **Add Signal(s)**.

In the Define MPC Structure By Linearization dialog box, the **Block Path** for the selected signal type updates.

Note If your model has measured disturbances, you must connect the corresponding plant inputs to the signal line connected to the md port of the MPC Controller block. For more information, see "Connect Measured Disturbances for Linearization" on page 2-50.

Specify Operating Point

In the **Simulink Operating Point** section, in the drop-down list, select an operating point at which to linearize the model.

For information on the different operating point options, see "Specifying Operating Points" on page 2-39.

Note If you select an option that generates multiple operating points for linearization, **MPC Designer** uses only the first operating point to define the plant structure and linearize the model.

Define Structure and Linearize Model

Click **Define and Linearize**.

The app linearizes the Simulink model at the specified operating point using the specified input/output signals, and adds the linearized plant to the **Data Browser**.
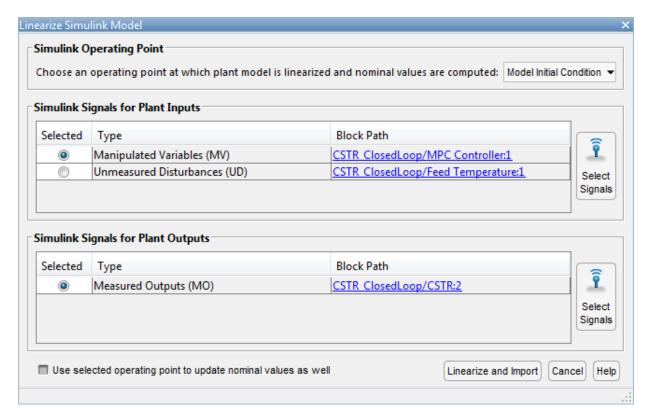
Also, a default controller, which uses the linearized plant as its internal model, and a default simulation scenario are created.

MPC Designer uses the input/output signal values at the selected operating point as nominal values.

Linearize Model

After you define the initial MPC structure, you can linearize the Simulink model at different operating points and import the linearized plants. Doing so is useful for validating controller performance against modeling errors.

On the **MPC Designer** tab, in the **Import** section, click **Linearize Model**.

Select Plant Input/Output Signals

In the **Simulink Signals for Plant Inputs** and **Simulink Signals for Plant Outputs** sections, the input/output signal configuration is the same as you specified when initially defining the MPC structure.

You cannot change the signal types and dimensions once the structure has been defined. However, for each signal type, you can select different signal lines from your Simulink model. The selected lines must have the same dimensions as defined in the current MPC structure.

Specify Operating Point

In the **Simulink Operating Point** section, in the drop-down list, select the operating points at which to linearize the model.

For information on the different operating point options, see "Specifying Operating Points" on page 2-39.

Linearize Model and Import Plant

Click **Linearize and Import**.

MPC Designer linearizes the Simulink model at the defined operating point using the specified input/output signals, and adds the linearized plant to the **Data Browser**.

If you select the **Use selected operating point to update nominal values as well** option, the app updates the controller nominal values using the operating point signal values.
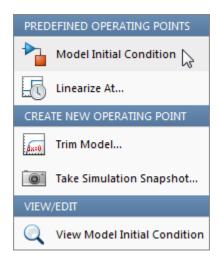
If you select an option that generates multiple operating points for linearization, the app linearizes the model at all the specified operating points. The linearized plants are added to the **Data Browser** in the same order in which their corresponding operating points are defined. If you choose to update the nominal values, the app uses the signal values from the first operating point.

Specifying Operating Points

In the **Simulink Operating Point** section, in the drop-down list, you can select or create operating points for model linearization. For more information on finding steady-state operating points, see "About Operating Points" (Simulink Control Design) and "Compute Steady-State Operating Points from Specifications" (Simulink Control Design).

Select Model Initial Condition
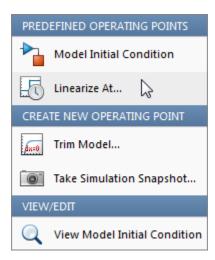
To linearize the model using the initial conditions specified in the Simulink model as the operating point, select **Model Initial Condition**.
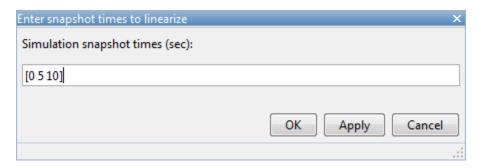
The model initial condition is the default operating point for linearization in **MPC Designer**.

Linearize at Simulation Snapshot Times

To linearize the model at specified simulation snapshot times, select **Linearize At**. Linearizing at snapshot times is useful when you know that your model reaches an equilibrium state after a certain simulation time.

In the Enter snapshot times to linearize dialog box, in the **Simulation snapshot times** field, enter one or more simulation snapshot times. Enter multiple snapshot times as a vector.
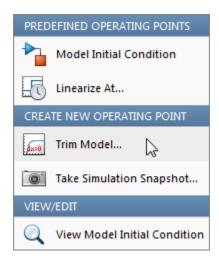
Click **OK**.

If you enter multiple snapshot times, and you selected **Linearize At** from the:
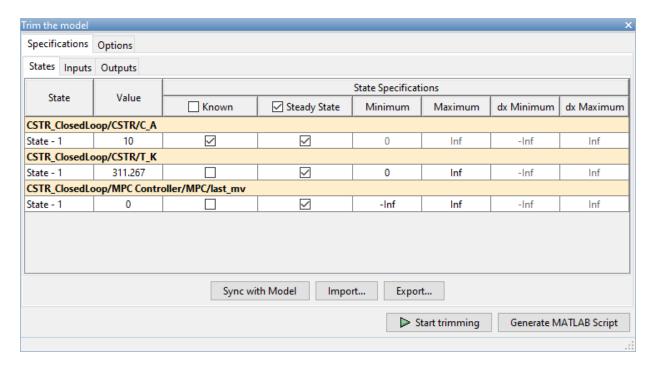
- Define MPC Structure By Linearization dialog box, **MPC Designer** linearizes the model using only the first snapshot time. The nominal values of the MPC controller are defined using the input/output signal values for this snapshot.
- Linearize Simulink Model dialog box, **MPC Designer** linearizes the model at all the specified snapshot times. The linearized plant models are added to the **Data Browser** in the order specified in the snapshot time array. If you selected the **Use selected operating point to update nominal values as well** option, the nominal values are set using the input/output signal values from the first snapshot.

Compute Steady-State Operating Point

To compute a steady-state operating point using numerical optimization methods to meet your specifications, select **Trim Model**.

In the Trim the model dialog box, enter the specifications for the steady-state values at which you want to find an operating point. You can specify values for states, input signals, and output signals.

Trim the model ✕

Specifications | Options

States | Inputs | Outputs

State	Value	State Specifications						
		☐ Known	☑ Steady State	Minimum	Maximum	dx Minimum	dx Maximum
CSTR_ClosedLoop/CSTR/C_A							
State - 1	10	☑	☑	0	Inf	-Inf	Inf
CSTR_ClosedLoop/CSTR/T_K							
State - 1	311.267	☐	☑	0	Inf	-Inf	Inf
CSTR_ClosedLoop/MPC Controller/MPC/last_mv							
State - 1	0	☐	☑	-Inf	Inf	-Inf	Inf

Sync with Model | Import... | Export...

▷ Start trimming | Generate MATLAB Script

Click **Start Trimming**.

MPC Designer creates an operating point for the given specifications. The computed operating point is added to the **Simulink Operating Point** drop-down list and is selected.
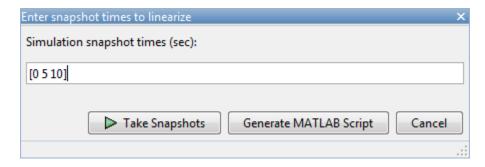
For examples showing how to specify the conditions for a steady-state operating point search, see "Compute Steady-State Operating Points from Specifications" (Simulink Control Design).

Compute Operating Point at Simulation Snapshot Time

To compute operating points using simulation snapshots, select **Take Simulation Snapshot**. Linearizing the model using operating points computed from simulation snapshots is useful when you know that your model reaches an equilibrium state after a certain simulation time.

In the Enter snapshot times to linearize dialog box, in the **Simulation snapshot times** field, enter one or more simulation snapshot times. Enter multiple snapshot times as a vector.

Click **Take Snapshots**.

MPC Designer simulates the Simulink model. At each snapshot time, the current state of the model is used to create an operating point, which is added to the drop-down list and selected.
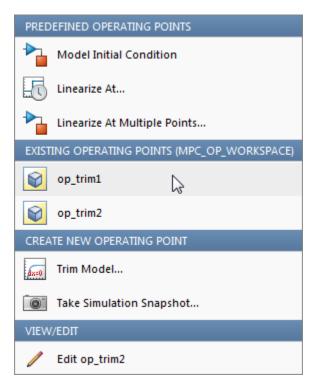
If you entered multiple snapshot times, the operating points are stored together as an array. If you selected **Take Simulation Snapshot** from the:

- Define MPC Structure By Linearization dialog box, **MPC Designer** linearizes the model using only the first operating point in the array. The nominal values of the MPC controller are defined using the input/output signal values for this operating point.
- Linearize Simulink Model dialog box, **MPC Designer** linearizes the model at all the operating points in the array. The linearized plant models are added to the **Data Browser** in the same order as the operating point array.

In **MPC Designer**, the **Linearize At** and **Take Simulation Snapshot** options generally produce the same linearized plant and nominal signal values. However, since the **Take Simulation Snapshot** option first computes an operating point from the snapshot before linearization, the results can differ.

Select Existing Operating Point

Under **Existing Operating Points**, select a previously defined operating point at which to linearize the Simulink model. This option is available if one or more previously created operating points are available in the drop-down list.
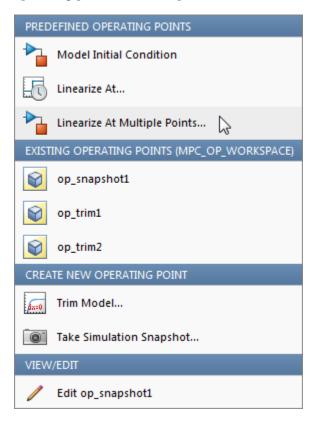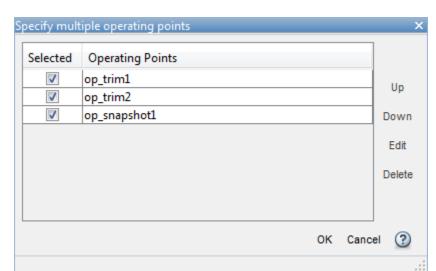
If the selected operating point represents an operating point array created using multiple snapshot times, and you selected an operating point from the:

- Define MPC Structure By Linearization dialog box, **MPC Designer** linearizes the model using only the first operating point in the array. The nominal values of the MPC controller are defined using the input/output signal values for this operating point.
- Linearize Simulink Model dialog box, **MPC Designer** linearizes the model at all the operating points in the array. The linearized plant models are added to the **Data Browser** in the same order as the operating point array.

Select Multiple Operating Points

To linearize the Simulink model at multiple existing operating points, select **Linearize at Multiple Points**. This option is available if there are more than one previously created operating points in the drop-down list.

In the Specify multiple operating points dialog box, select the operating points at which to linearize the model.

To change the operating point order, click an operating point in the list and click **Up** or **Down** to move the highlighted operating point within the list.
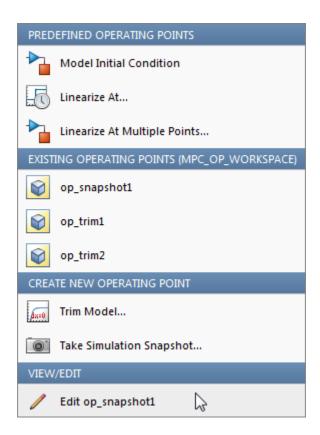
Click **OK**.

If you selected **Linearize at Multiple Points** from the:
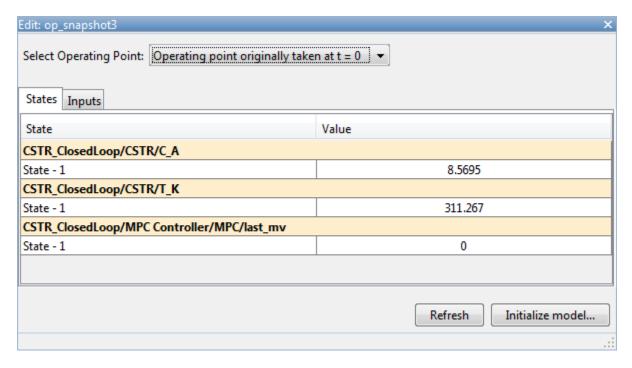
- Define MPC Structure By Linearization dialog box, **MPC Designer** linearizes the model using only the first specified operating point. The nominal values of the MPC controller are defined using the input/output signal values for this operating point.
- Linearize Simulink Model dialog box, **MPC Designer** linearizes the model at all the specified operating points. The linearized plant models are added to the **Data Browser** in the order specified in the Specify multiple operating points dialog box.

View/Edit Operating Point

To view or edit the selected operating point, under **View/Edit**, click the **Edit** option.
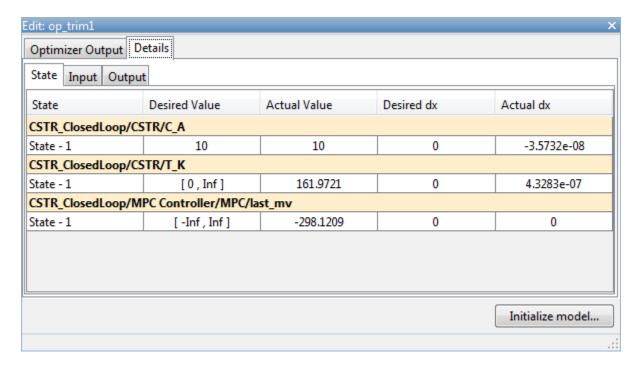
In the Edit dialog box, if you created the selected operating point from a simulation snapshot, you can edit the operating point values.

If the selected operating point represents an operating point array, in the **Select Operating Point** drop-down list, select an operating point to view.

If you obtained the operating point by trimming the model, you can only view the operating point values.

To set the Simulink model initial conditions to the states in the operating point, click **Initialize model**. You can then simulate the model at the specified operating point.

When setting the model initial conditions, **MPC Designer** exports the operating point to the MATLAB workspace. Also, in the Simulink Configuration Parameters dialog box, in the **Data Import/Export** section, it selects the **Input** and **Initial state** parameters and configures them to use the states and inputs in the exported operating point.
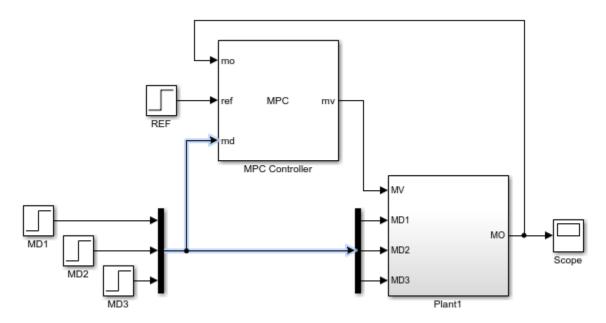
To reset the model initial conditions, for example if you delete the exported operating point, clear the **Input** and **Initial state** parameters.

Connect Measured Disturbances for Linearization

If your Simulink model has measured disturbance signals, connect them to the corresponding plant input ports and to the md port of the MPC Controller block. If you have multiple measured disturbances, connect them to the MPC Controller using a vector signal. As discussed in "Define MPC Structure By Linearization" on page 2-32, **MPC**
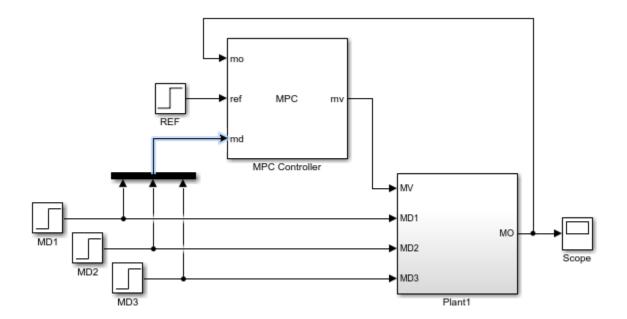
Designer automatically detects the measured disturbances connected to the MPC Controller block and sets them as plant inputs for linearization.

Since the measured disturbances connected to the md port are selected as linearization inputs, you must connect the plant measured disturbance input ports to the selected signal line, as shown in the following:

Correct MD Connection

If you connect the plant measured disturbance input ports to the corresponding signals before the Mux block, as shown in the following, there is no linearization path from the signals at the md port to the plant. As a result, when you linearize the plant using **MPC Designer**, the measured disturbance channels linearize to zero.

Incorrect MD Connection

See Also

MPC Designer

Related Examples

Identify Plant from Data

When designing a model predictive controller, you can specify the internal predictive plant model using a linear identified model. You use System Identification Toolbox software to estimate a linear plant model in one of these forms:

- State-space model — `idss`
- Transfer function model — `idtf`
- Polynomial model — `idpoly`
- Process model — `idproc`
- Grey-box model — `idgrey`

You can estimate the plant model programmatically at the command line or interactively using the **System Identification** app.

Identify Plant from Data at the Command Line

This example shows how to identify a plant model at the command line. For information on identifying models using the System Identification app, see "Identify Linear Models Using System Identification App" (System Identification Toolbox).

Load the measured input/output data.

```
load plantIO
```

This command imports the plant input signal, `u`, plant output signal, `y`, and sample time, `Ts` to the MATLAB® workspace.

Create an `iddata` object from the input and output data.

```
mydata = iddata(y,u,Ts);
```

You can optionally assign channel names and units for the input and output signals.

```
mydata.InputName = 'Voltage';
mydata.InputUnit = 'V';
mydata.OutputName = 'Position';
mydata.OutputUnit = 'cm';
```

Typically, you must preprocess identification I/O data before estimating a model. For this example, remove the offsets from the input and output signals by detrending the data.

```
mydatad = detrend(mydata);
```

You can also remove offsets by creating an `ssestOptions` object and specifying the `InputOffset` and `OutputOffset` options.

For this example, estimate a second-order, linear state-space model using the detrended data. To estimate a discrete-time model, specify the sample time as `Ts`.

```
ss1 = ssest(mydatad,2,'Ts',Ts)

ss1 =
  Discrete-time identified state-space model:
    x(t+Ts) = A x(t) + B u(t) + K e(t)
        y(t) = C x(t) + D u(t) + e(t)

  A =
             x1        x2
    x1    0.8942   -0.1575
    x2    0.1961    0.7616

  B =
          Voltage
    x1  6.008e-05
    x2   -0.01219

  C =
                 x1        x2
    Position    38.24   -0.3835

  D =
            Voltage
    Position       0

  K =
        Position
    x1   0.03572
    x2    0.0223

Sample time: 0.1 seconds

Parameterization:
   FREE form (all coefficients in A, B, C free).
   Feedthrough: none
   Disturbance component: estimate
   Number of free coefficients: 10
```

```
   Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using SSEST on time domain data "mydatad".
Fit to estimation data: 89.85% (prediction focus)
FPE: 0.0156, MSE: 0.01541
```

You can use this identified plant as the internal prediction model for your MPC controller. When you do so, the controller converts the identified model to a discrete-time, state-space model.

By default, the MPC controller discards any unmeasured noise components from your identified model. To configure noise channels as unmeasured disturbances, you must first create an augmented state-space model from your identified model. For example:

```
ss2 = ss(ss1,'augmented')

ss2 =

  A =
            x1         x2
   x1    0.8942   -0.1575
   x2    0.1961    0.7616

  B =
         Voltage   v@Position
   x1   6.008e-05    0.004448
   x2    -0.01219    0.002777

  C =
                 x1         x2
   Position    38.24   -0.3835

  D =
              Voltage   v@Position
   Position          0       0.1245

Input groups:
     Name       Channels
   Measured        1
    Noise          2

Sample time: 0.1 seconds
Discrete-time state-space model.
```

This command creates a state-space model, `ss2`, with two input groups, `Measured` and `Noise`, for the measured and noise inputs respectively. When you import the augmented model into your MPC controller, channels in the `Noise` input group are defined as unmeasured disturbances.

Working with Impulse-Response Models

You can use System Identification Toolbox software to estimate finite step-response or finite impulse-response (FIR) plant models using measured data. Such models, also known as nonparametric models, are easy to determine from plant data ([1] and [2]) and have intuitive appeal.

Use the `impulseest` function to estimate an FIR model from measured data. This function generates the FIR coefficients encapsulated as an `idtf` object; that is, a transfer function model with only numerator coefficients. `impulseest` is especially effective in situations where the input signal used for identification has low excitation levels. To design a model predictive controller for this plant, you can convert the identified FIR plant model to a numeric LTI model. However, this conversion usually yields a high-order plant, which can degrade the controller design. For example, the numerical precision issues with high-order plants can affect estimator design. This result is particularly an issue for MIMO systems.

Model predictive controllers work best with low-order parametric models. Therefore, to design a model predictive controller using measured plant data, you can:

*   Estimate a low-order parametric model using a parametric estimator, such as `ssest`.

*   Initially identify a nonparametric model using `impulseest`, and then estimate a low-order parametric model from the response of the nonparametric model. For an example, see [3].

*   Initially identify a nonparametric model using `impulseest`, and then convert the FIR model to a state-space model using `idss`. You can then reduce the order of the state-space model using `balred`. This approach is similar to the method used by `ssregest`.

References

[1] Cutler, C., and F. Yocum, "Experience with the DMC inverse for identification," *Chemical Process Control — CPC IV* (Y. Arkun and W. H. Ray, eds.), CACHE, 1991.

[2] Ricker, N. L., "The use of bias least-squares estimators for parameters in discrete-time pulse response models," *Ind. Eng. Chem. Res.*, Vol. 27, pp. 343, 1988.

[3] Wang, L., P. Gawthrop, C. Chessari, T. Podsiadly, and A. Giles, "Indirect approach to continuous time system identification of food extruder," *J. Process Control*, Vol. 14, Number 6, pp. 603–615, 2004.

See Also

Apps
System Identification

Functions
`detrend` | `iddata` | `ssest`

More About

- "Handling Offsets and Trends in Data" (System Identification Toolbox)
- "Identify Linear Models Using System Identification App" (System Identification Toolbox)
- "Design MPC Controller for Identified Plant Model"

3

Design MPC Controllers

Design Controller Using MPC Designer

This example shows how to design a model predictive controller for a continuous stirred-tank reactor (CSTR) using **MPC Designer**.

CSTR Model

The following differential equations represent the linearized model of a continuous stirred-tank reactor (CSTR) involving an exothermic reaction:

$$\frac{dC'_A}{dt} = a_{11}C'_A + a_{12}T' + b_{11}T'_c + b_{12}C'_{Ai}$$

$$\frac{dT'}{dt} = a_{21}C'_A + a_{22}T' + b_{21}T'_c + b_{22}C'_{Ai}$$
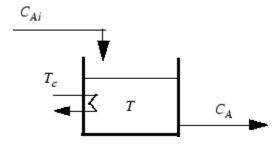
where the inputs are:

- C_{Ai} — Concentration of reagent A in the feed stream (kgmol/m^3)
- T_c — Reactor coolant temperature (degrees C)

and the outputs are:

- T — Reactor temperature (degrees C)
- C_A — Residual concentration of reagent A in the product stream (kgmol/m^3)

The prime terms, such as C'_A, denote a deviation from the nominal steady-state condition at which the model has been linearized.

Measurement of reagent concentrations is often difficult. For this example, assume that:

- T_c is a manipulated variable.

- C_{Ai} is an unmeasured disturbance.
- T is a measured output.
- C_A is an unmeasured output.

The model can be described in state-space format:

$$\frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

where,

$$x = \begin{bmatrix} C'_A \\ T' \end{bmatrix}, \ u = \begin{bmatrix} T'_c \\ C'_{Ai} \end{bmatrix}, \ y = \begin{bmatrix} T' \\ C'_A \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \ B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \ C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \ D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

For this example, the coolant temperature has a limited range of ±10 degrees from its nominal value and a limited rate of change of ±4 degrees per sample period.

Create a state-space model of a CSTR system.

```
A = [-0.0285 -0.0014; -0.0371 -0.1476];
B = [-0.0850 0.0238; 0.0802 0.4462];
C = [0 1; 1 0];
D = zeros(2,2);
CSTR = ss(A,B,C,D);
```
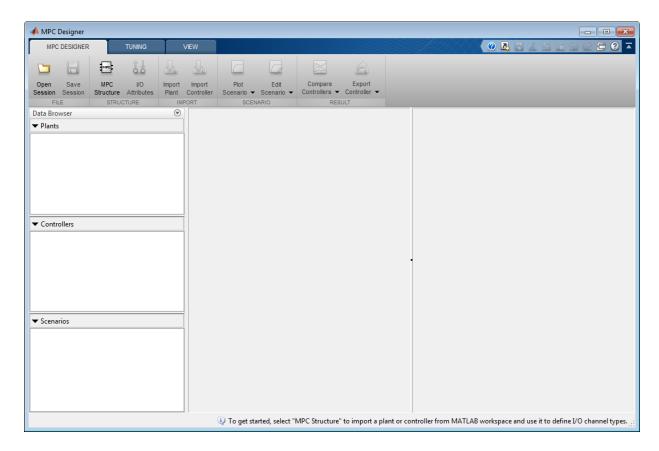
Import Plant and Define MPC Structure

```
mpcDesigner
```

On the **MPC Designer** tab, in the **Structure** section, click **MPC Structure**.
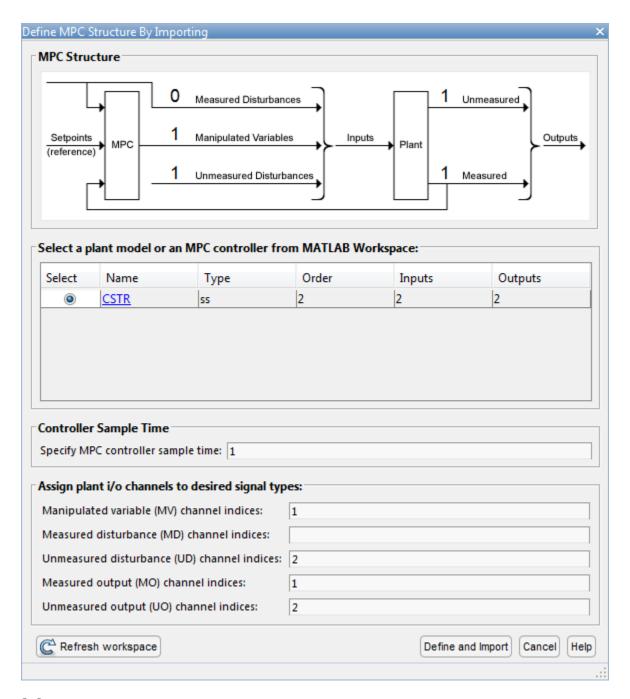
In the Define MPC Structure By Importing dialog box, in the **Select a plant model or an MPC controller** table, select the CSTR model.

Since CSTR is a stable, continuous-time LTI system, **MPC Designer** sets the controller sample time to 0.1 T_r, where T_r is the average rise time of CSTR. For this example, in the **Specify MPC controller sample time** field, enter a sample time of 1.

By default, all plant inputs are defined as manipulated variables and all plant outputs as measured outputs. In the **Assign plant i/o channels** section, assign the input and output channel indices such that:

• The first input, coolant temperature, is a manipulated variable.

- The second input, feed concentration, is an unmeasured disturbance.
- The first output, reactor temperature, is a measured output.
- The second output, reactant concentration, is an unmeasured output.

Define MPC Structure By Importing ✕

MPC Structure

Select a plant model or an MPC controller from MATLAB Workspace:

Select	Name	Type	Order	Inputs	Outputs
◉	CSTR	ss	2	2	2

Controller Sample Time

Specify MPC controller sample time: 1

Assign plant i/o channels to desired signal types:

Manipulated variable (MV) channel indices: 1

Measured disturbance (MD) channel indices:

Unmeasured disturbance (UD) channel indices: 2

Measured output (MO) channel indices: 1

Unmeasured output (UO) channel indices: 2

Refresh workspace   Define and Import   Cancel   Help

Click **Define and Import**.

The app imports the CSTR plant to the **Data Browser**. The following are also added to the **Data Browser**:

- mpc1 — Default MPC controller created using sys as its internal model.
- scenario1 — Default simulation scenario.

The app runs the default simulation scenario and updates the **Input Response** and **Output Response** plots.

Tip To view the response plots side-by-side, on the **View** tab, in the **Tiles** section, click **Left/Right**.

Once you define the MPC structure, you cannot change it within the current **MPC Designer** session. To use a different channel configuration, start a new session of the app.

Define Input and Output Channel Attributes
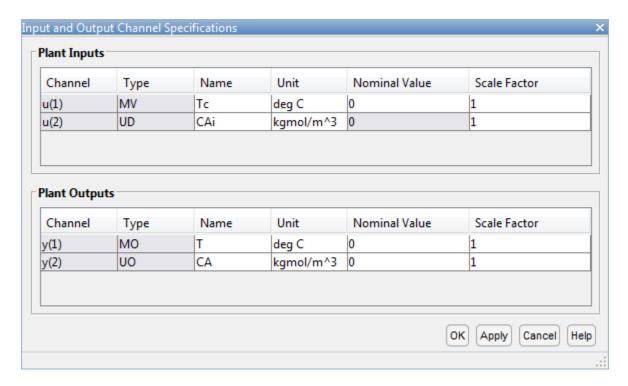
On the **MPC Designer** tab, select **I/O Attributes**.

In the Input and Output Channel Specifications dialog box, in the **Name** column, specify a meaningful name for each input and output channel.

In the **Unit** column, optionally specify the units for each channel.

Since the state-space model is defined using deviations from the nominal operating point, set the **Nominal Value** for each input and output channel to 0.

Keep the **Scale Factor** for each channel at the default value of 1.

Click **OK**.

The **Input Response** and **Output Response** plot labels update to reflect the new signal names and units.
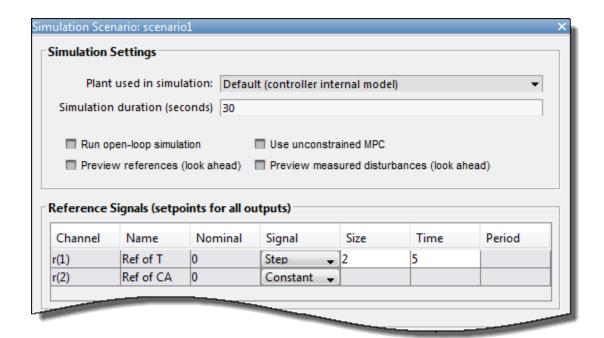
Configure Simulation Scenario

On the **MPC Designer** tab, in the **Scenario** section, click **Edit Scenario > scenario1**.

In the Simulation Scenario dialog box, increase the **Simulation duration** to 30 seconds.

In the **Reference Signals** table, in the first row, specify a step **Size** of 2 and a **Time** of 5.

In the **Signal** column, in the second row, select a Constant reference to hold the concentration setpoint at its nominal value.
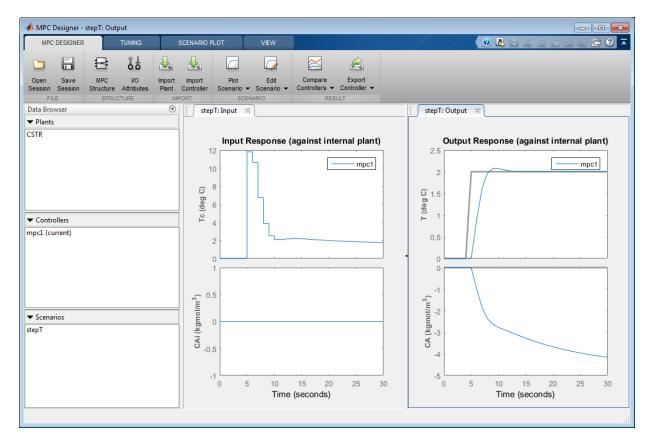
The default scenario is configured to simulate a step change of 2 degrees in the reactor temperature, *T*, at a time of 5 seconds.

Click **OK**.
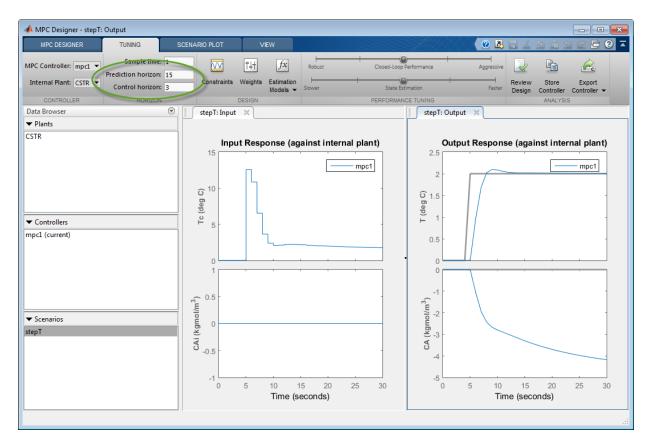
The response plots update to reflect the new simulation scenario configuration.

In the **Data Browser**, in the **Scenarios** section, click scenario1. Click scenario1 a second time, and rename the scenario to stepT.

Configure Controller Horizons

On the **Tuning** tab, in the **Horizons** section, specify a **Prediction horizon** of 15 and a **Control horizon** of 3.
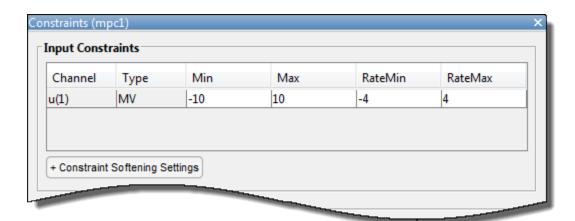
The response plots update to reflect the new horizons. The **Input Response** plot shows that the control actions for the manipulated variable violate the required coolant temperature constraints.

Define Input Constraints

In the **Design** section, click **Constraints**.
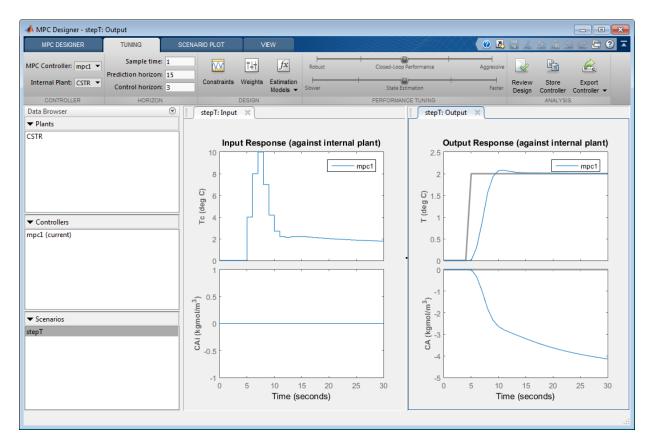
In the Constraints dialog box, in the **Input Constraints** section, enter the coolant temperature upper and lower bounds in the **Min** and **Max** columns respectively.

Specify the rate of change limits in the **RateMin** and **RateMax** columns.

Click **OK**.

The **Input Response** plot shows the constrained manipulated variable control actions. Even with the constrained rate of change, the coolant temperature rises quickly to its maximum limit within three control intervals.
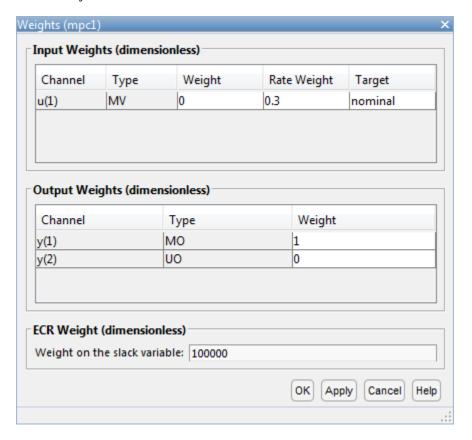
Specify Controller Tuning Weights

On the **Tuning** tab, in the **Design** section, click **Weights**.
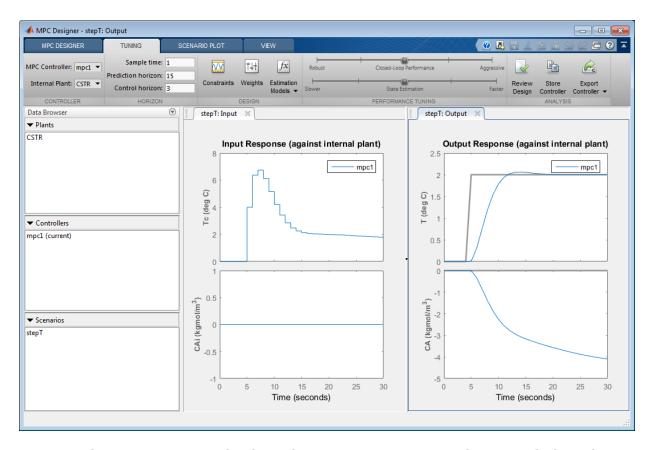
In the **Input Weights** table, increase the manipulated variable (MV) **Rate Weight** to `0.3`. Increasing the MV rate weight penalizes large MV changes in the controller optimization cost function.

In the **Output Weights** table, keep the default **Weight** values. By default, all unmeasured outputs have zero weights.

Since there is only one manipulated variable, if the controller tries to hold both outputs at specific setpoints, one or both outputs will exhibit steady-state error in their responses. Since the controller ignores setpoints for outputs with zero weight, setting the concentration output weight to zero allows reactor temperature setpoint tracking with zero steady-state error.

Weights (mpc1)   ✕

Input Weights (dimensionless)

Channel	Type	Weight	Rate Weight	Target
u(1)	MV	0	0.3	nominal

Output Weights (dimensionless)

Channel	Type	Weight
y(1)	MO	1
y(2)	UO	0

ECR Weight (dimensionless)

Weight on the slack variable: 100000

[OK] [Apply] [Cancel] [Help]

Click **OK**.
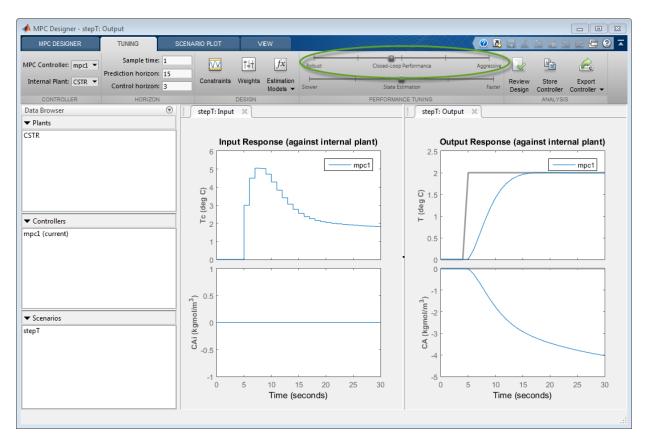
The **Input Response** plot shows the more conservative control actions, which result in a slower **Output Response**.

Eliminate Output Overshoot

Suppose the application demands zero overshoot in the output response. On the **Performance Tuning** tab, drag the **Closed-Loop Performance** slider to the left until the **Output Response** has no overshoot. Moving this slider to the left simultaneously increases the manipulated variable rate weight of the controller and decreases the output variable weight, producing a more robust controller.

When you adjust the controller tuning weights using the **Closed-Loop Performance** slider, **MPC Designer** does not change the weights you specified in the Weights dialog box. Instead, the slider controls an adjustment factor, which is used with the user-specified weights to define the actual controller weights.

This factor is `1` when the slider is centered; its value decreases as the slider moves left and increases as the slider moves right. The weighting factor multiplies the manipulated variable and output variable weights and divides the manipulated variable rate weights from the Weights dialog box.

To view the actual controller weights, export the controller to the MATLAB workspace, and view the `Weights` property of the exported controller object.

Test Controller Disturbance Rejection
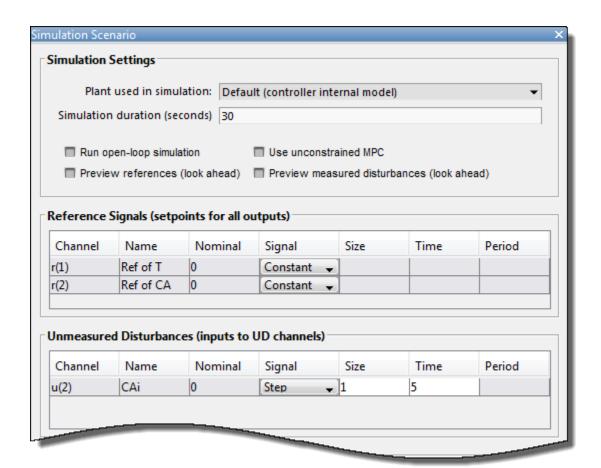
In a process control application, disturbance rejection is often more important than setpoint tracking. Simulate the controller response to a step change in the feed concentration unmeasured disturbance.

On the **MPC Designer** tab, in the **Scenario** section, click **Plot Scenario > New Scenario**.

In the Simulation Scenario dialog box, set the **Simulation duration** to 30 seconds.

In the **Unmeasured Disturbances** table, in the **Signal** drop-down list, select Step.
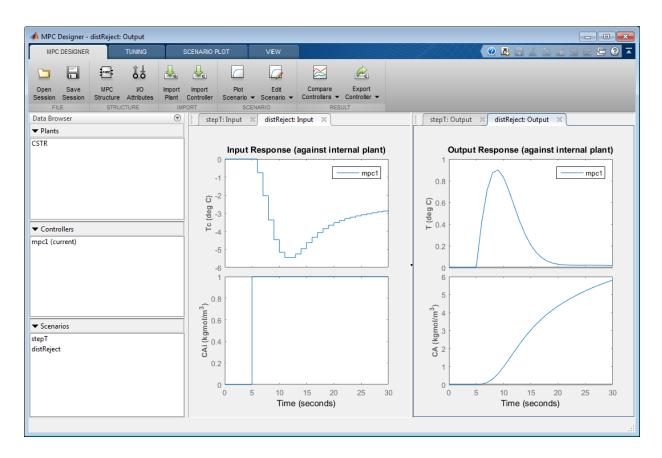
In the **Time** column, specify a step time of 5 seconds.

Click **OK**.

The app adds new scenario to the **Data Browser** and creates new corresponding **Input Response** and **Output Response** plots.

In the **Data Browser**, in the **Scenarios** section, rename NewScenario to distReject.

In the **Output Response** plots, the controller returns the reactor temperature, **T**, to a value near its setpoint as expected. However, the required control actions cause an increase in the output concentration, **CA** to 6 kgmol/m³.

Specify Concentration Output Constraint
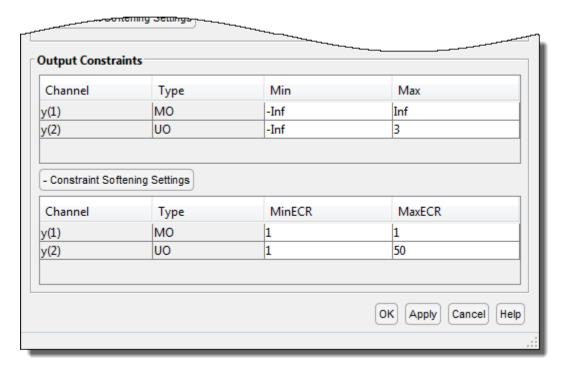
Previously, you defined the controller tuning weights to achieve the primary control objective of tracking the reactor temperature setpoint with zero steady-state error. Doing so enables the unmeasured reactor concentration to vary freely. Suppose that unwanted reactions occur once the reactor concentration exceeds a 3 kgmol/m³. To limit the reactor concentration, specify an output constraint.

On the **Tuning** tab, in the **Design** section, click **Constraints**.
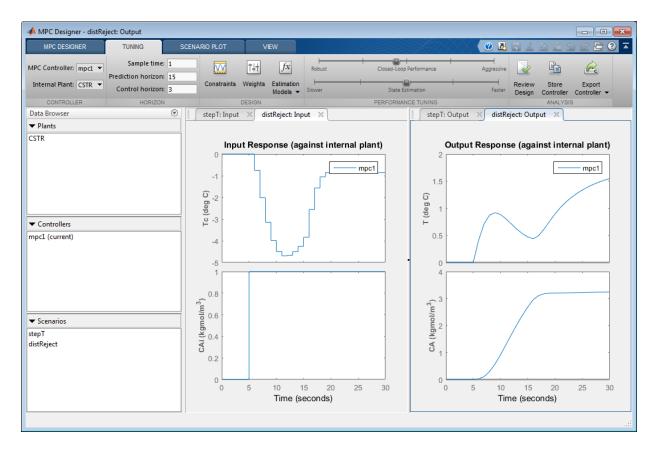
In the Constraints dialog box, in the **Output Constraints** section, the second row of the table, specify a **Max** unmeasured output (UO) value of 3.

In the **Output Constraints** section, click **Constraint Softening Settings**.

By default, all output constraints are soft, meaning that their **MinECR** and **MaxECR** values are greater than zero. To soften the unmeasured output (UO) constraint further, increase its **MaxECR** value.

Output Constraints

Channel	Type	Min	Max
y(1)	MO	-Inf	Inf
y(2)	UO	-Inf	3

- Constraint Softening Settings

Channel	Type	MinECR	MaxECR
y(1)	MO	1	1
y(2)	UO	1	50

OK   Apply   Cancel   Help

Click **OK**.

In the **Output Response** plots, once the reactor concentration, **CA**, approaches 3 kgmol/m^3, the reactor temperature, **T**, starts to increase. Since there is only one manipulated variable, the controller makes a compromise between the two competing control objectives: Temperature control and constraint satisfaction. A softer output constraint enables the controller to sacrifice the constraint requirement more to achieve improved temperature tracking.

Since the output constraint is soft, the controller maintain adequate temperature control by allowing a small concentration constraint violation. In general, depending on your application requirements, you can experiment with different constraint settings to achieve an acceptable control objective compromise.

Export Controller

In the **Tuning** tab, in the **Analysis** section, click **Export Controller** to save the tuned controller, `mpc1`, to the MATLAB workspace.

Delete Plants, Controllers, and Scenarios

To delete a plant, controller, or scenario, in the **Data Browser**, right-click the item you want to delete, and select **Delete**. You can also click the item and hit **Delete** on the keyboard.

You cannot delete the current controller. Also, you cannot delete a plant or scenario if it is the only listed plant or scenario.

If a plant is used by any controller or scenario, you cannot delete the plant.

To delete multiple plants, controllers, or scenarios, hold **Shift** and click each item that you want to delete.

References

[1] Seborg, D. E., T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*, 2nd Edition, Wiley, 2004, pp. 34–36 and 94–95.

See Also
MPC Designer

More About
- "Specify Constraints"
- "Tune Weights"
- "Design MPC Controller in Simulink" on page 3-38

Design MPC Controller at the Command Line

This example shows how to create and test a model predictive controller from the command line.

Define Plant Model

This example uses the plant model described in "Design Controller Using MPC Designer" on page 3-2. Create a state-space model of the plant and set some of the optional model properties.

```
A = [-0.0285 -0.0014; -0.0371 -0.1476];
B = [-0.0850 0.0238; 0.0802 0.4462];
C = [0 1; 1 0];
D = zeros(2,2);
CSTR = ss(A,B,C,D);

CSTR.InputName = {'T_c','C_A_i'};
CSTR.OutputName = {'T','C_A'};
CSTR.StateName = {'C_A','T'};
CSTR.InputGroup.MV = 1;
CSTR.InputGroup.UD = 2;
CSTR.OutputGroup.MO = 1;
CSTR.OutputGroup.UO = 2;
```

Create Controller

To improve the clarity of the example, suppress Command Window messages from the MPC controller.

```
old_status = mpcverbosity('off');
```

Create a model predictive controller with a control interval, or sample time, of 1 second, and with all other properties at their default values.

```
Ts = 1;
MPCobj = mpc(CSTR,Ts)


MPC object (created on 26-Aug-2019 22:28:46):
---------------------------------------------
Sampling time:      1 (seconds)
Prediction Horizon: 10
Control Horizon:    2
```

```
Plant Model:
                                -------------
       1  manipulated variable(s)   -->|  2 states   |
                                |             |--> 1 measured output(s)
       0  measured disturbance(s)   -->|  2 inputs   |
                                |             |--> 1 unmeasured output(s)
       1  unmeasured disturbance(s) -->|  2 outputs  |
                                -------------
Indices:
  (input vector)     Manipulated variables: [1 ]
                   Unmeasured disturbances: [2 ]
  (output vector)          Measured outputs: [1 ]
                        Unmeasured outputs: [2 ]

Disturbance and Noise Models:
        Output disturbance model: default (type "getoutdist(MPCobj)" for details)
         Input disturbance model: default (type "getindist(MPCobj)" for details)
         Measurement noise model: default (unity gain after scaling)

Weights:
         ManipulatedVariables: 0
     ManipulatedVariablesRate: 0.1000
              OutputVariables: [1 0]
                          ECR: 100000

State Estimation:  Default Kalman Filter (type "getEstimator(MPCobj)" for details)

Unconstrained
```

View and Modify Controller Properties

Display a list of the controller properties and their current values.

```
get(MPCobj)

                          Ts: 1
      PredictionHorizon (P): 10
         ControlHorizon (C): 2
                       Model: [1x1 struct]
  ManipulatedVariables (MV): [1x1 struct]
      OutputVariables (OV): [1x2 struct]
  DisturbanceVariables (DV): [1x1 struct]
               Weights (W): [1x1 struct]
                  Optimizer: [1x1 struct]
```

3-25

```
                    Notes: {}
                 UserData: []
                  History: 26-Aug-2020 22:28:46
```

The displayed `History` value will be different for your controller, since it depends on when the controller was created. For a description of the editable properties of an MPC controller, enter `mpcprops` at the command line.

Use dot notation to modify these properties. For example, change the prediction horizon to 15.

```
MPCobj.PredictionHorizon = 15;
```

You can abbreviate property names provided that the abbreviation is unambiguous.

Many of the controller properties are structures containing additional fields. Use dot notation to view and modify these field values. For example, you can set the measurement units for the controller output variables. The `OutputUnit` property is for display purposes only and is optional.

```
MPCobj.Model.Plant.OutputUnit = {'Deg C','kmol/m^3'};
```

By default, the controller has no constraints on manipulated variables and output variables. You can view and modify these constraints using dot notation. For example, set constraints for the controller manipulated variable.

```
MPCobj.MV.Min = -10;
MPCobj.MV.Max = 10;
MPCobj.MV.RateMin = -3;
MPCobj.MV.RateMax = 3;
```

You can also view and modify the controller tuning weights. For example, modify the weights for the manipulated variable rate and the output variables.

```
MPCobj.W.ManipulatedVariablesRate = 0.3;
MPCobj.W.OutputVariables = [1 0];
```

You can also define time-varying constraints and weights over the prediction horizon, which shifts at each time step. Time-varying constraints have a nonlinear effect when they are active. For example, to force the manipulated variable to change more slowly towards the end of the prediction horizon, enter:

```
MPCobj.MV.RateMin = [-4; -3.5; -3; -2.5];
```

```
MPCobj.MV.RateMax = [4; 3.5; 3; 2.5];
```

The -2.5 and 2.5 values are used for the fourth step and beyond.

Similarly, you can specify different output variable weights for each step of the prediction horizon. For example, enter:
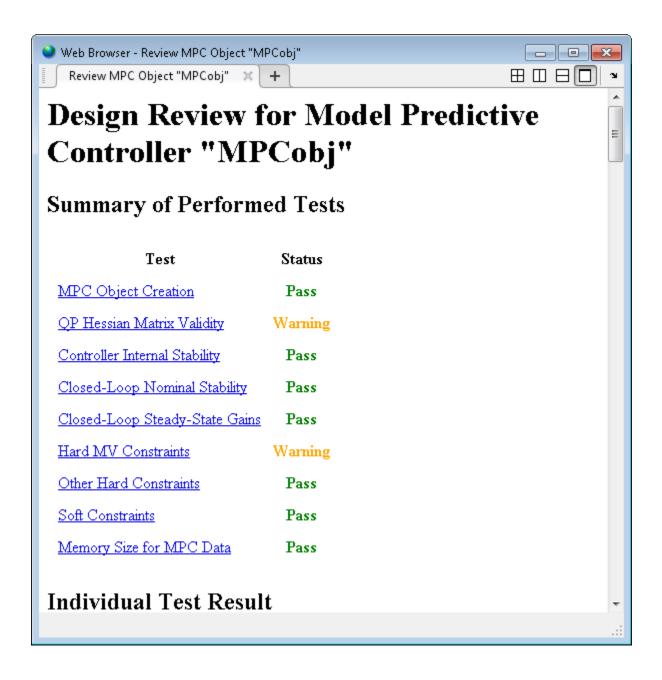
```
MPCobj.W.OutputVariables = [0.1 0; 0.2 0; 0.5 0; 1 0];
```

You can also modify the disturbance rejection characteristics of the controller. See setEstimator, setindist, and setoutdist for more information.

Review Controller Design

Generate a report on potential run-time stability and performance issues.

```
review(MPCobj)
```

Design Review for Model Predictive Controller "MPCobj"

Summary of Performed Tests

Test	Status
MPC Object Creation	Pass
QP Hessian Matrix Validity	Warning
Controller Internal Stability	Pass
Closed-Loop Nominal Stability	Pass
Closed-Loop Steady-State Gains	Pass
Hard MV Constraints	Warning
Other Hard Constraints	Pass
Soft Constraints	Pass
Memory Size for MPC Data	Pass

Individual Test Result

In this example, the `review` command found two potential issues with the design. The first warning asks whether the user intends to have a weight of zero on the `C_A` output. The second warning advises the user to avoid having hard constraints on both `MV` and `MVRate`.

Perform Linear Simulations

Use the `sim` function to run a linear simulation of the system. For example, simulate the closed-loop response of `MPCobj` for `26` control intervals. Specify setpoints of `2` and `0` for the reactor temperature and the residual concentration respectively. The setpoint for the residual concentration is ignored because the tuning weight for the second output is zero.
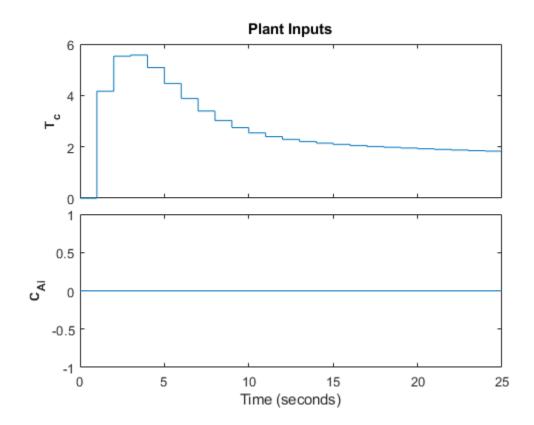
```
T = 26;
r = [0 0; 2 0];
sim(MPCobj,T,r)
```

Plant Outputs
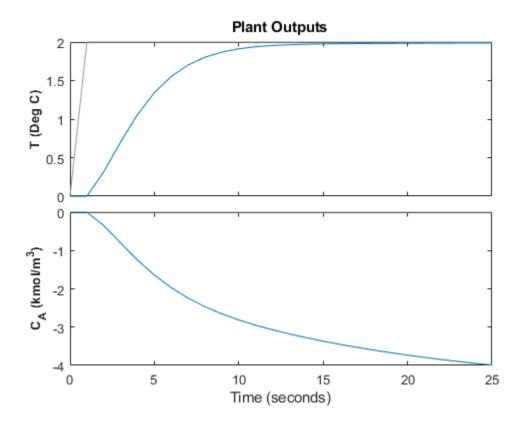
You can modify the simulation options using `mpcsimopt`. For example, run a simulation with the manipulated variable constraints turned off.

```
MPCopts = mpcsimopt;
MPCopts.Constraints = 'off';
sim(MPCobj,T,r,MPCopts)
```

Plant Outputs
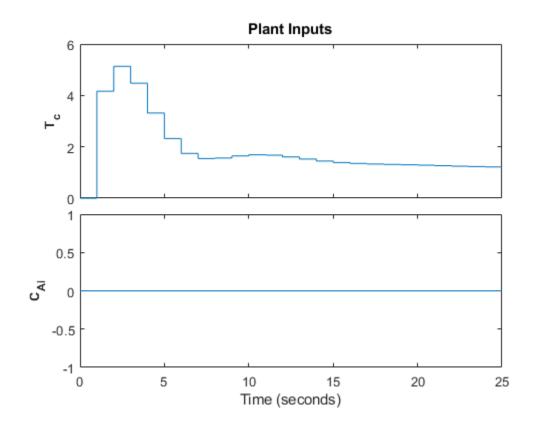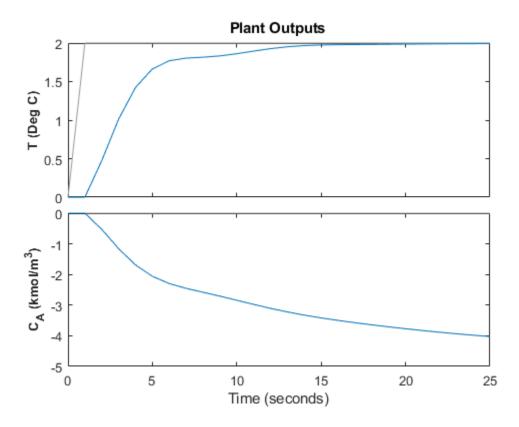
The first move of the manipulated variable now exceeds the specified 3-unit rate constraint.

You can also perform a simulation with a plant/model mismatch. For example, define a plant with 50% larger gains than those in the model used by the controller.

```
Plant = 1.5*CSTR;
MPCopts.Model = Plant;
sim(MPCobj,T,r,MPCopts)
```

The plant/model mismatch degrades controller performance slightly. Degradation can be severe and must be tested on a case-by-case basis.

Other options include the addition of a specified noise sequence to the manipulated variables or measured outputs, open-loop simulations, and a look-ahead option for better setpoint tracking or measured disturbance rejection.
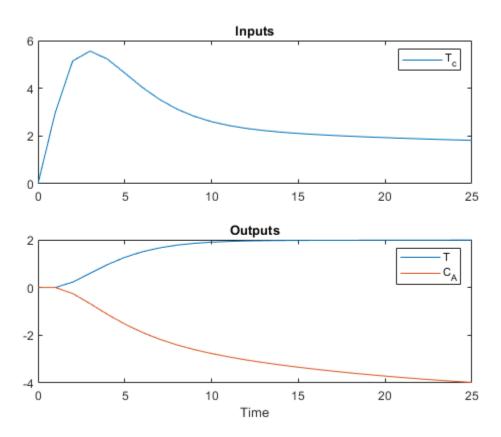
Store Simulation Results

Store the simulation results in the MATLAB Workspace.

```
[y,t,u] = sim(MPCobj,T,r);
```

3-35

The syntax suppresses automatic plotting and returns the simulation results. You can use the results for other tasks, including custom plotting. For example, plot the manipulated variable and both output variables in the same figure.

```
figure
subplot(2,1,1)
plot(t,u)
title('Inputs')
legend('T_c')
subplot(2,1,2)
plot(t,y)
title('Outputs')
legend('T','C_A')
xlabel('Time')
```

Restore the mpcverbosity setting.

```
mpcverbosity(old_status);
```

See Also
mpc | review | sim

More About
- "MPC Modeling" on page 2-2
- "Design Controller Using MPC Designer" on page 3-2
- "Design MPC Controller in Simulink" on page 3-38

3-37

Design MPC Controller in Simulink
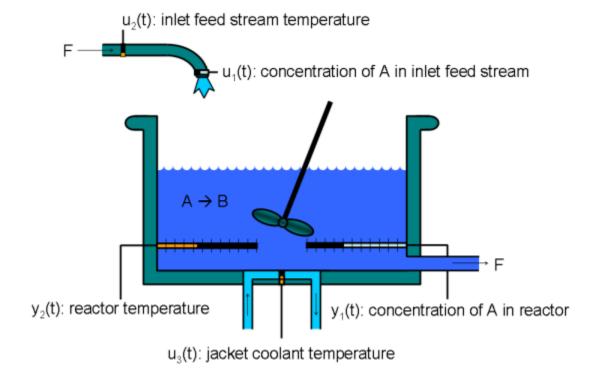
This example shows how to design a model predictive controller for a continuous stirred-tank reactor (CSTR) in Simulink using **MPC Designer**.

This example requires Simulink Control Design software to define the MPC structure by linearizing a nonlinear Simulink model.

If you do not have Simulink Control Design software, you must first create an `mpc` object in the MATLAB workspace. For more information, see "Design Controller Using MPC Designer" on page 3-2 and "Design MPC Controller at the Command Line" on page 3-24.

CSTR Model

A CSTR is a jacketed nonadiabatic tank reactor commonly used in the process industry.

An inlet stream of reagent *A* feeds into the tank at a constant rate. A first-order, irreversible, exothermic reaction takes place to produce the product stream, which exits the reactor at the same rate as the input stream.

The CSTR model has three inputs:

- Feed Concentration (*CAi*) — The concentration of reagent *A* in the feed stream (kgmol/m^3)
- Feed Temperature (*Ti*) — Feed stream temperature (K)
- Coolant Temperature (*Tc*) — Reactor coolant temperature (K)

The two model outputs are:
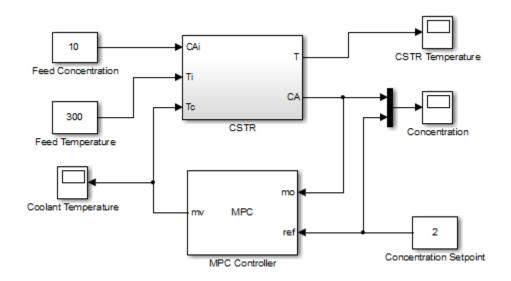
- CSTR Temperature (*T*) — Reactor temperature (K)
- Concentration (*CA*) — Concentration of reagent *A* in the product stream, also referred to as the residual concentration (kgmol/m^3)

The control objective is to maintain the residual concentration, *CA*, at its nominal setpoint by adjusting the coolant temperature, *Tc*. Changes in the feed concentration, *CAi*, and feed temperature, *Ti*, cause disturbances in the CSTR reaction.

The reactor temperature, *T*, is usually controlled. However, for this example, ignore the reactor temperature, and assume that the residual concentration is measured directly.

Open Simulink Model
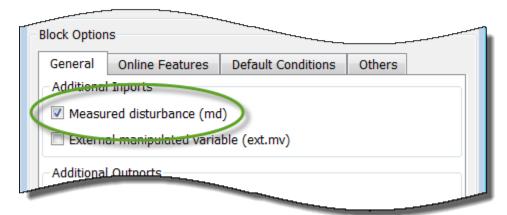
```
open_system('CSTR_ClosedLoop')
```

Connect Measured Disturbance To MPC Controller Block
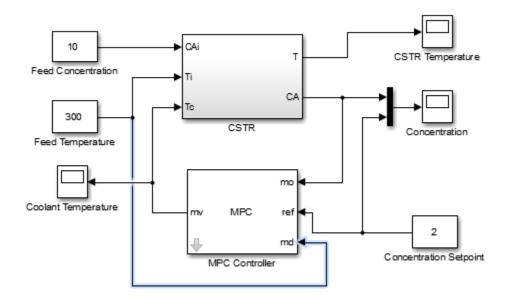
In the Simulink model window, double-click the MPC Controller block.

In the Block Parameters dialog box, on the **General** tab, in the **Additional Inports** section, check the **Measured disturbance (md)** option.

Click **Apply** to add the md inport to the controller block.

In the Simulink model window, connect the Feed Temperature block output to the md inport.

Open MPC Designer App

In the MPC Controller Block Parameters dialog box, click **Design** to open **MPC Designer**.

Note This step requires Simulink Control Design software to linearize the Simulink model. For more information, see "Linearize Simulink Models Using MPC Designer" on page 2-32.

If you do not have Simulink Control Design software, you must first create an mpc object in the MATLAB workspace.
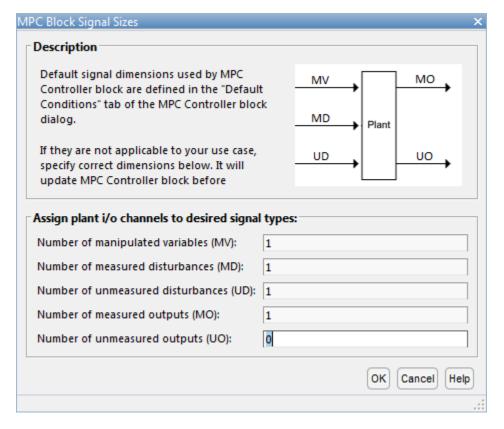
Define MPC Structure

In **MPC Designer**, on the **MPC Designer** tab, in the **Structure** section, click **MPC Structure**.

In the Define MPC Structure By Linearization dialog box, in the **Controller Sample Time** section, specify a sample time of `0.1`.
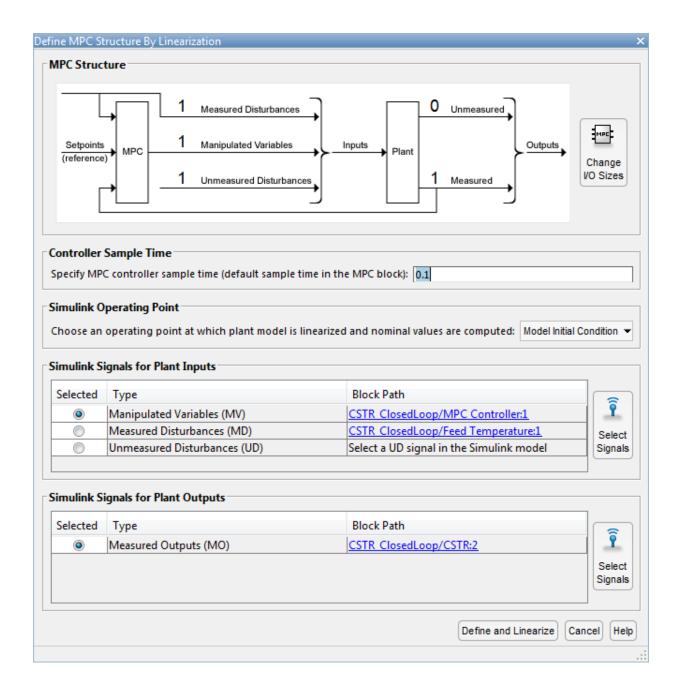
In the **MPC Structure** section, click **Change I/O Sizes** to add the unmeasured disturbance and measured disturbance signal dimensions.

In the MPC Block Signal Sizes dialog box, specify the number of input/output channels of each type:

Click **OK**.

In the Define MPC Structure By Linearization dialog box, in the **Simulink Signals for Plant Inputs** section, the app adds a row for **Unmeasured Disturbances (UD)**.
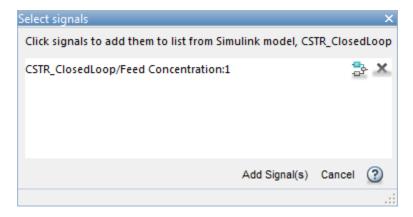
The manipulated variable, measured disturbance, and measured output are already assigned to their respective Simulink signal lines, which are connected to the MPC Controller block.

In the **Simulink Signals for Plant Inputs** section, select the **Unmeasured Disturbances (UD)** row, and click **Select Signals**.

In the Simulink model window, click the output signal from the Feed Concentration block.

The signal is highlighted and its block path is added to the Select Signal dialog box.
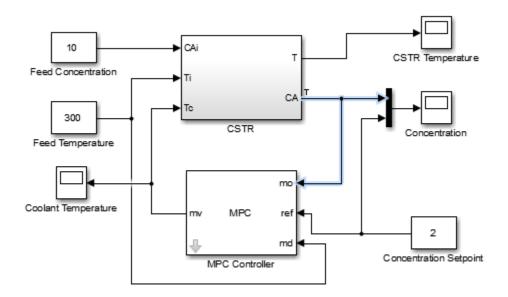
In the Select Signals dialog box, click **Add Signal(s)**.

In the Define MPC Structure By Linearization dialog box, in the **Simulink Signals for Plant Inputs** table, the **Block Path** for the unmeasured disturbance signal is updated.

Linearize Simulink Model

Linearize the Simulink model at a steady-state equilibrium operating point where the residual concentration is 2 kgmol/m^3. To compute such an operating point, add the CA signal as a trim output constraint, and specify its target constraint value.
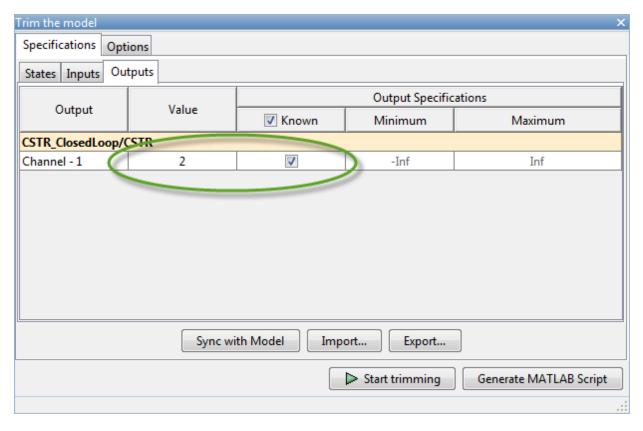
In the Simulink model window, right-click the signal line connected to CA outport of the CSTR block, and select **Linear Analysis Points > Trim Output Constraint**.

The CA signal can now be used to define output specifications for calculating a model steady-state operating point.

In the Define MPC Structure By Linearization dialog box, in the **Simulink Operating Point** section, in the drop-down list, under **Create New Operating Point**, click **Trim Model**.

In the Trim the model dialog box, in the **Outputs** tab, check the box in the **Known** column for Channel-1 and specify a **Value** of 2.
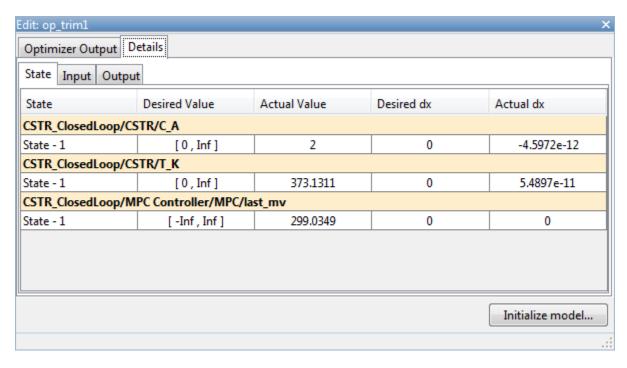
This setting constrains the value of the output signal during the operating point search to a known value.

Click **Start Trimming**.

In the Define MPC Structure By Linearization dialog box, in the **Simulink Operating Point** section, the computed operating point, `op_trim1`, is added to the drop-down list and selected.

In the drop-down list, under **View/Edit**, click **Edit op_trim1**.

Edit: op_trim1				×

Optimizer Output | **Details**

State | Input | Output

State	Desired Value	Actual Value	Desired dx	Actual dx
CSTR_ClosedLoop/CSTR/C_A				
State - 1	[0 , Inf]	2	0	-4.5972e-12
CSTR_ClosedLoop/CSTR/T_K				
State - 1	[0 , Inf]	373.1311	0	5.4897e-11
CSTR_ClosedLoop/MPC Controller/MPC/last_mv				
State - 1	[-Inf , Inf]	299.0349	0	0

Initialize model...

In the Edit dialog box, on the **State** tab, in the **Actual dx** column, the near-zero derivative values indicate that the computed operating point is at steady-state.

Click **Initialize model** to set the initial states of the Simulink model to the operating point values in the **Actual Values** column. Doing so enables you to later simulate the Simulink model at the computed operating point rather than at the default model initial conditions.

In the Initialize Model dialog box, click **OK**.

When setting the model initial conditions, **MPC Designer** exports the operating point to the MATLAB workspace. Also, in the Simulink Configuration Parameters dialog box, in the **Data Import/Export** section, it selects the **Input** and **Initial state** parameters and configures them to use the states and inputs in the exported operating point.

To reset the model initial conditions, for example if you delete the exported operating point, clear the **Input** and **Initial state** parameters.

Close the Edit dialog box.

In the Define MPC Structure By Linearization dialog box, click **Define and Linearize** to linearize the model.

The linearized plant model is added to the **Data Browser**. Also, the following are added to the **Data Browser**:
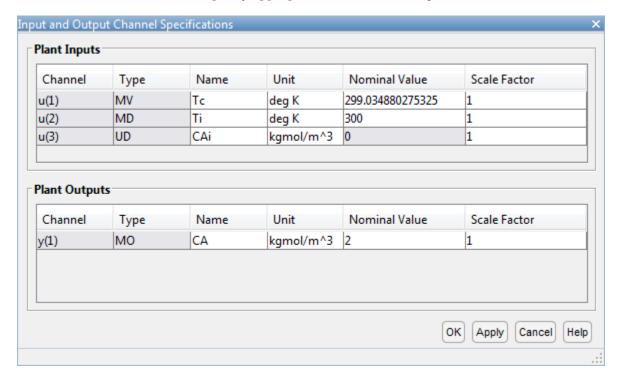
- A default MPC controller created using the linearized plant as an internal prediction model
- A default simulation scenario

Define Input/Output Channel Attributes

On the **MPC Designer** tab, in the **Structure** section, click **I/O Attributes**.

In the Input and Output Channel Specifications dialog box, in the **Name** column, specify meaningful names for each input and output channel.

In the **Unit** column, specify appropriate units for each signal.

Input and Output Channel Specifications

Plant Inputs

Channel	Type	Name	Unit	Nominal Value	Scale Factor
u(1)	MV	Tc	deg K	299.034880275325	1
u(2)	MD	Ti	deg K	300	1
u(3)	UD	CAi	kgmol/m^3	0	1

Plant Outputs

Channel	Type	Name	Unit	Nominal Value	Scale Factor
y(1)	MO	CA	kgmol/m^3	2	1

OK   Apply   Cancel   Help

The **Nominal Value** for each signal is the corresponding steady-state value at the computed operating point.

Click **OK**.

Define Disturbance Rejection Simulation Scenarios
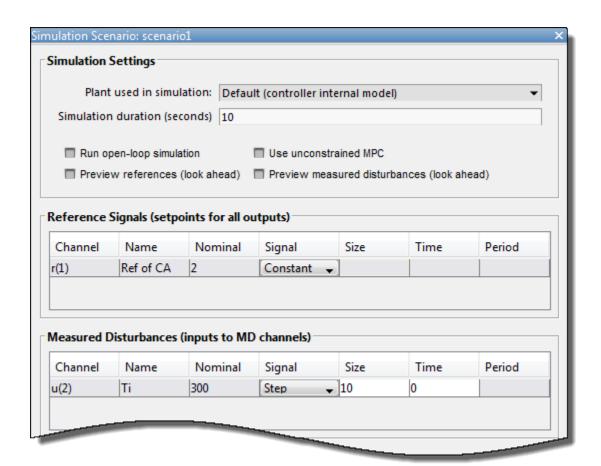
The primary objective of the controller is to hold the residual concentration, *CA*, at the nominal value of 2 kgmol/m^3. To do so, the controller must reject both measured and unmeasured disturbances.

In the **Scenario** section, click **Edit Scenario > scenario1**.

In the Simulation Scenario dialog box, in the **Reference Signals** table, in the **Signal** drop-down list select **Constant** to hold the output setpoint at its nominal value.

In the **Measured Disturbances** table, in the **Signal** drop-down list, select **Step**.

Specify a step **Size** of 10 and a step **Time** of 0.

Click **OK**.
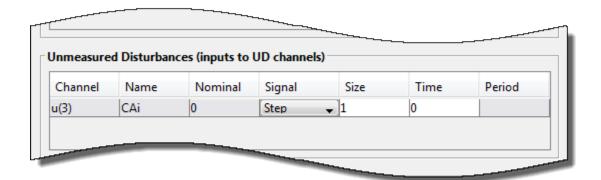
In the **Data Browser**, under **Scenarios**, click scenario1. Click scenario1 a second time, and rename it MD_reject.

In the **Scenario** section, click **Plot Scenario > New Scenario**.

In the Simulation Scenario dialog box, in the **Unmeasured Disturbances** table, in the **Signal** drop-down list, select **Step**.

Specify a step **Size** of 1 and a step **Time** of 0.

Click **OK**.

In the **Data Browser**, under **Scenarios**, rename NewScenario to UD_reject.
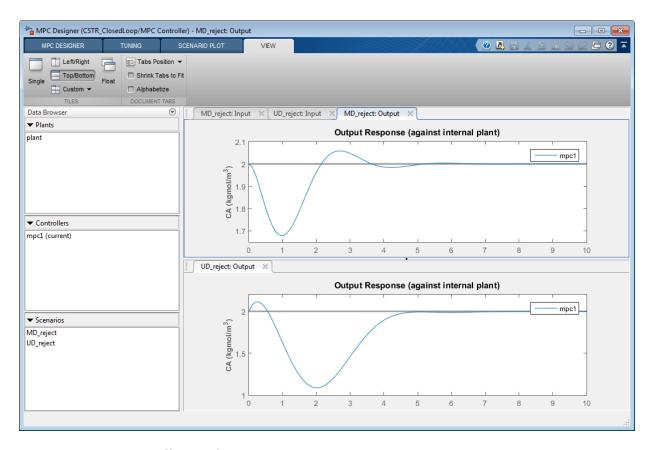
Arrange Output Response Plots

To make viewing the tuning results easier, arrange the plot area to display the Output Response plots for both scenarios at the same time.

On the **View** tab, in the **Tiles** section, click **Top/Bottom**.

The plot display area changes to display the Input Response plots above the Output Response plots.

Drag the **MD_reject: Output** tab up to the top plot.
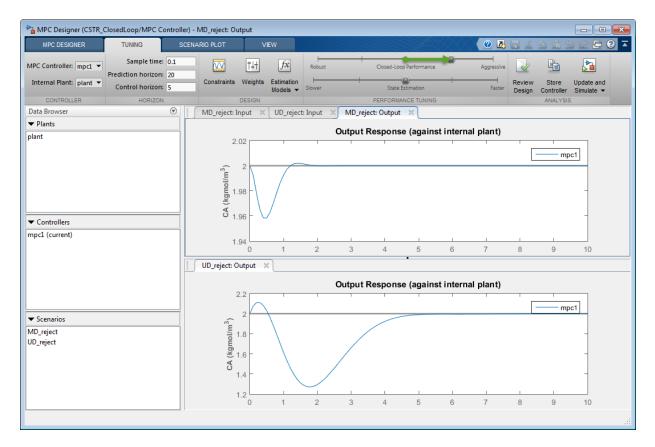
Tune Controller Performance

In the **Tuning** tab, in the **Horizon** section, specify a **Prediction horizon** of 20 and a **Control horizon** of 5.
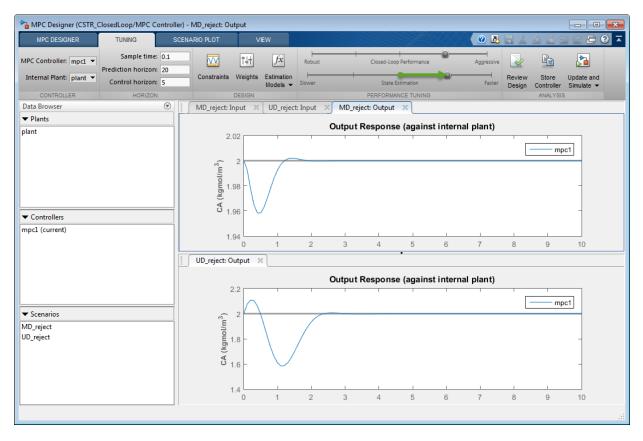
The **Output Response** plots update based on the new horizon values.

Use the default controller constraint and weight configurations.

In the **Performance Tuning** section, drag the **Closed-Loop Performance** slider to the right, which leads to tighter control of outputs and more aggressive control moves. Drag the slider until the **MD_reject: Output** response reaches steady state in less than 2 seconds.

Drag the **State Estimation** slider to the right, which leads to more aggressive unmeasured disturbance rejection. Drag the slider until the **UD_reject: Output** response reaches steady state in less than 3 seconds.
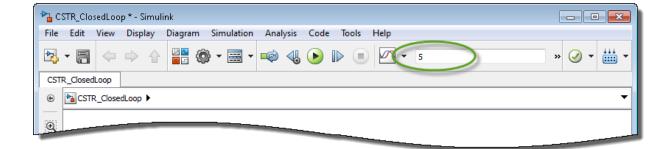
Update Simulink Model with Tuned Controller

In the **Analysis** section, click the **Update and Simulate** arrow ▾.

Under **Update and Simulate**, click **Update Block Only**. The app exports the tuned controller, mpc1, to the MATLAB workspace. In the Simulink model, the MPC Controller block is updated to use the exported controller.

Simulate Unmeasured Disturbance Rejection
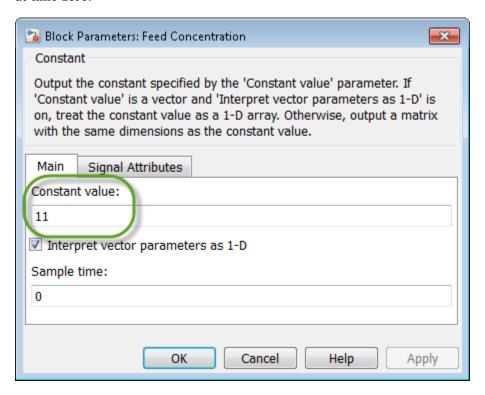
In the Simulink model window, change the simulation duration to 5 seconds.

The model initial conditions are set to the nominal operating point used for linearization.
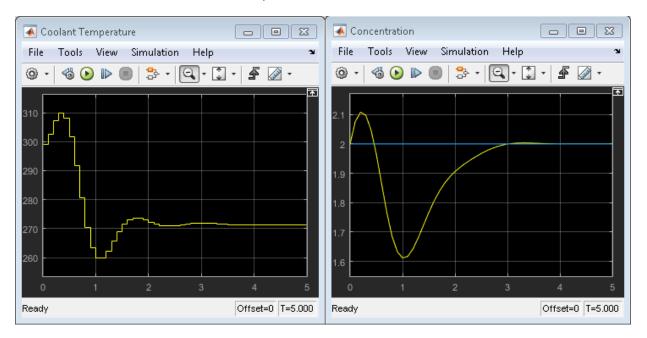
Double-click the **Feed Concentration** block.

In the Block Parameters dialog box, enter a **Constant Value** of 11 to simulate a unit step at time zero.

Click **Apply**.

In the Simulink model window, click **Run**.

The **Concentration** output response is similar to the **UD_reject** response, however the settling time is around 1 second later. The different result is due to the mismatch between the linear plant used in the **MPC Designer** simulation and the nonlinear plant in the Simulink model.

Simulate Measured Disturbance Rejection
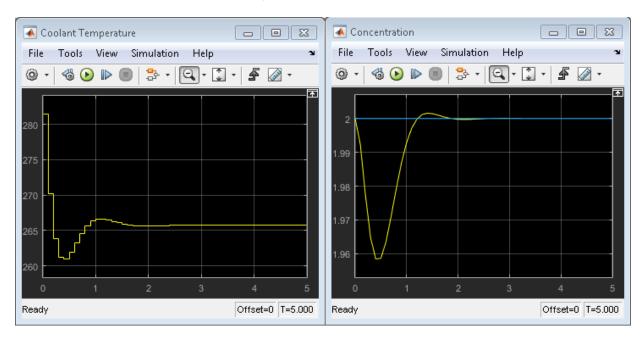
In the Block Parameters: Feed Concentration dialog box, enter a **Constant Value** of 10 to return the feed concentration to its nominal value.

Click **OK**.

In the Simulink model window, double-click the Feed Temperature block.

In the Block Parameters: Feed Temperature dialog box, enter a **Constant Value** of 310 to simulate a step change of size 10 at time zero.

Click **OK**.

In the Simulink model window, click **Run**.

The **Concentration** output response is similar to the **MD_reject** response from the **MPC Designer** simulation.

References

[1] Seborg, D. E., T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*, 2nd Edition, Wiley, 2004, pp. 34–36 and 94–95.

See Also

MPC Controller | **MPC Designer**

More About

- "Tune Weights"
- "Linearize Simulink Models" on page 2-22

- "Design Controller Using MPC Designer" on page 3-2

Control of a Single-Input-Single-Output Plant

This example shows how to control a double integrator plant under input saturation in Simulink®.

Define Plant Model

The linear open-loop dynamic model is a double integrator.

```
plant = tf(1,[1 0 0]);
```

Design MPC Controller

Create the controller object with sampling period, prediction and control horizons.

```
Ts = 0.1;
p = 10;
m = 3;
mpcobj = mpc(plant, Ts, p, m);
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming c
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
```

Specify actuator saturation limits as MV constraints.

```
mpcobj.MV = struct('Min',-1,'Max',1);
```

Simulate Using Simulink
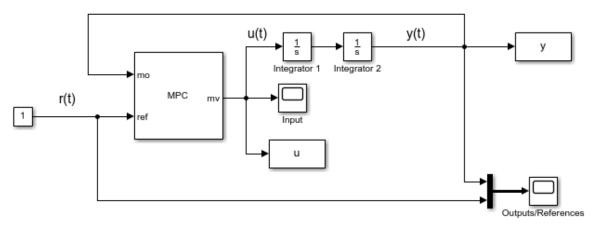
To run this example, Simulink is required.

```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink is required to run this example.')
    return
end
```
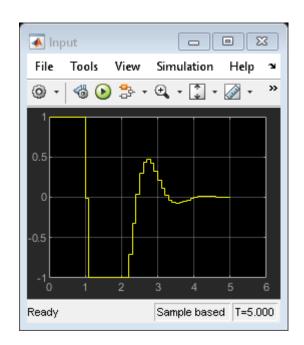
Simulate closed-loop control of the linear plant model in Simulink. The MPC Controller block is configured to use the mpcobj object as its controller.
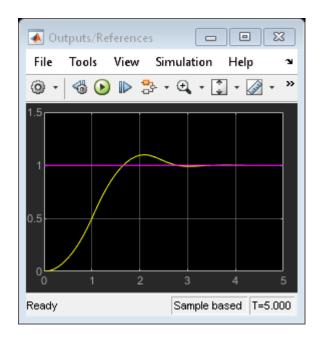
```
mdl = 'mpc_doubleint';
open_system(mdl)
sim(mdl)
```

```
-->Converting the "Model.Plant" property of "mpc" object to state-space.
-->Converting model to discrete time.
```

Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac

Copyright 1990-2014 The MathWorks, Inc.

The closed-loop response shows good setpoint tracking performance.

```
bdclose(mdl)
```

See Also

MPC Controller | **MPC Designer** | mpc

More About

Control of a Multi-Input Single-Output Plant

This example shows how to design model predictive controller with one measured output, one manipulated variable, one measured disturbance, and one unmeasured disturbance in a typical workflow.

Define Plant Model

The discrete-time linear open-loop dynamic model is defined below with sample time Ts.

```
sys = ss(tf({1,1,1},{[1 .5 1],[1 1],[.7 .5 1]}));
Ts = 0.2;
model = c2d(sys,Ts);
```

Design MPC Controller

Define type of input signals: the first signal is a manipulated variable, the second signal is a measured disturbance, the third one is an unmeasured disturbance.

```
model = setmpcsignals(model,'MV',1,'MD',2,'UD',3);
```

Create the controller object with sampling period, prediction and control horizons.

```
mpcobj = mpc(model,Ts,10,3);
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
```

Define constraints on the manipulated variable.

```
mpcobj.MV = struct('Min',0,'Max',1,'RateMin',-10,'RateMax',10);
```

For unmeasured input disturbances, its model is an integrator driven by white noise with variance = 1000.

```
mpcobj.Model.Disturbance = tf(sqrt(1000),[1 0]);
```

Simulate Closed-Loop Response Using the sim Command
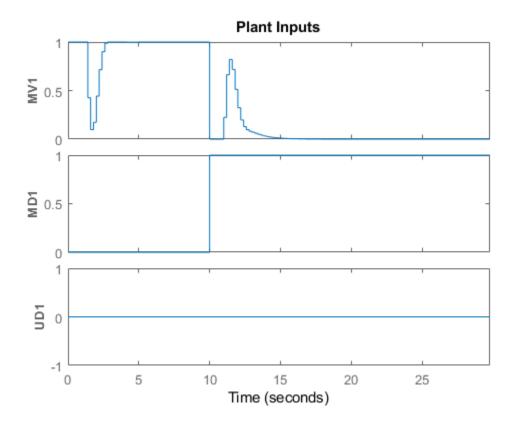
Specify simulation parameters.

```
Tstop = 30;                              % simulation time
Tf = round(Tstop/Ts);                    % number of simulation steps
```
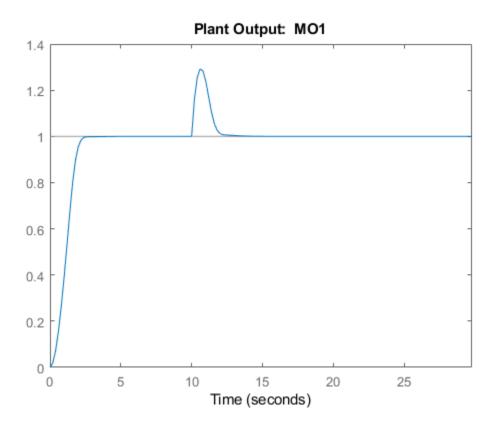
```
r = ones(Tf,1);                        % reference signal
v = [zeros(Tf/3,1);ones(2*Tf/3,1)];    % measured disturbance signal
```

Run the closed-loop simulation and plot results.

```
sim(mpcobj,Tf,r,v)
```

```
    Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

Plant Output: MO1

Specify disturbance and noise signals in simulation option object.
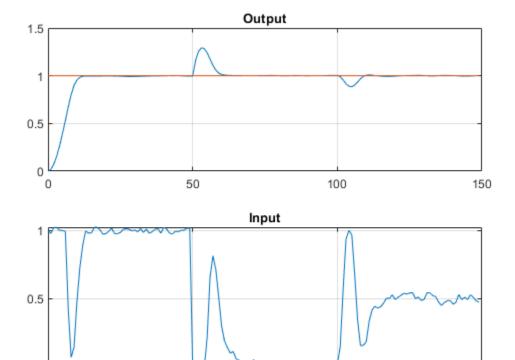
```
SimOptions = mpcsimopt(mpcobj);
d = [zeros(2*Tf/3,1);-0.5*ones(Tf/3,1)];
SimOptions.Unmeas = d;                          % unmeasured input disturbance signal
SimOptions.OutputNoise=.001*(rand(Tf,1)-.5);    % output measurement noise
SimOptions.InputNoise=.05*(rand(Tf,1)-.5);      % noise on manipulated variables
```

Run the closed-loop simulation and save the results to the workspace.

```
[y,t,u,xp] = sim(mpcobj,Tf,r,v,SimOptions);
```

Plot results.

```
figure
subplot(2,1,1)
plot(0:Tf-1,y,0:Tf-1,r)
title('Output')
grid
subplot(2,1,2)
plot(0:Tf-1,u)
title('Input')
grid
```


Simulate Closed-Loop Response with Model Mismatch

Test the robustness of the MPC controller against a model mismatch. Specify the true plant `simModel`, with a nominal output value of `0.1`.
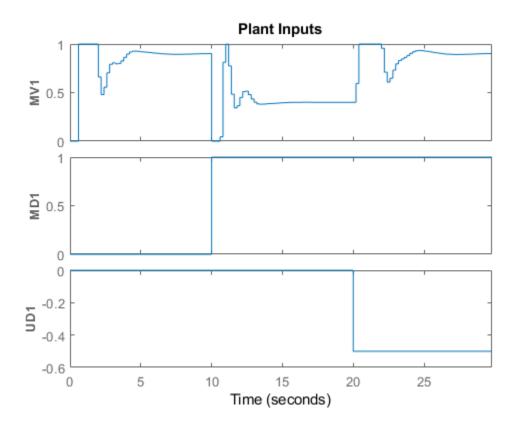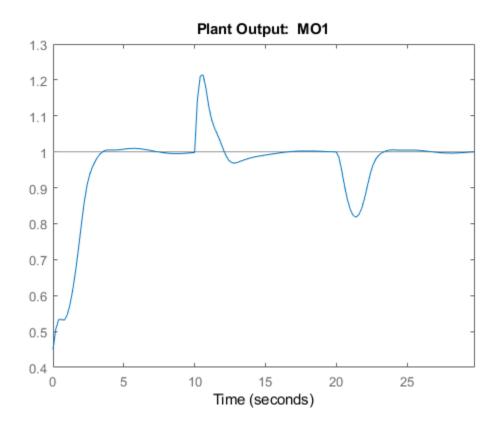
```matlab
simModel = ss(tf({1,1,1},{[1 .8 1],[1 2],[.6 .6 1]}));
simModel = setmpcsignals(simModel,'MV',1,'MD',2,'UD',3);
simModel = struct('Plant',simModel);
simModel.Nominal.Y = 0.1;
simModel.Nominal.X = -.1*[1 1 1 1 1];
```

Create option object.

```matlab
SimOptions.Model = simModel;
SimOptions.plantinit = [0.1 0 -0.1 0 .05];  % Initial state of the true plant
SimOptions.OutputNoise = [];                % remove output measurement noise
SimOptions.InputNoise = [];                 % remove noise on manipulated variables

% Run the closed-loop simulation and plot results.
sim(mpcobj,Tf,r,v,SimOptions)

-->Converting model to discrete time.
```

Plant Output: MO1

Soften Constraints

Relax the constraints on manipulated variables from hard to soft.

```
mpcobj.MV.MinECR = 1;
mpcobj.MV.MaxECR = 1;
```
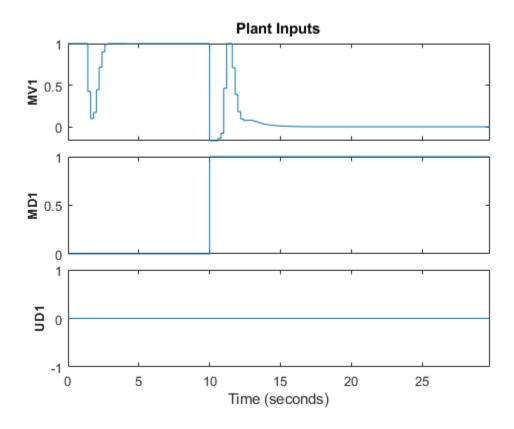
Define an output constraint.
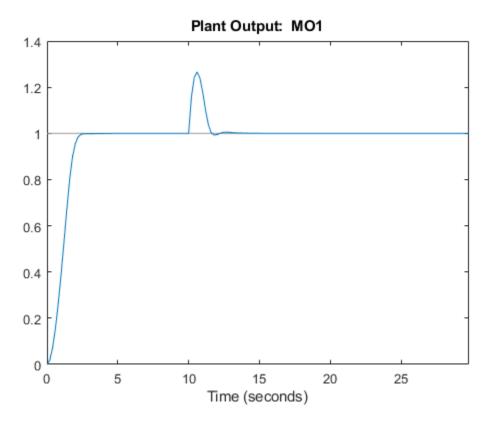
```
mpcobj.OV.Max = 1.1;
```

Run a new closed-loop simulation.

```
sim(mpcobj,Tf,r,v)
```

Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac

Plant Output: MO1
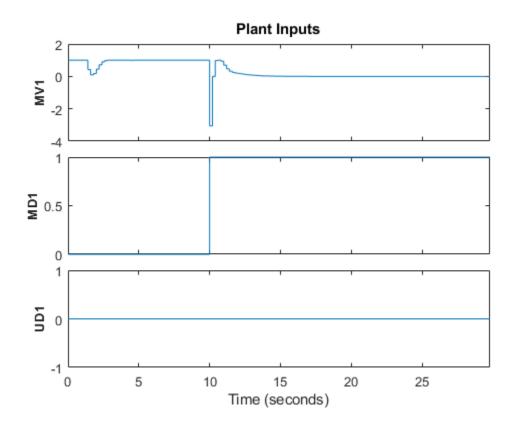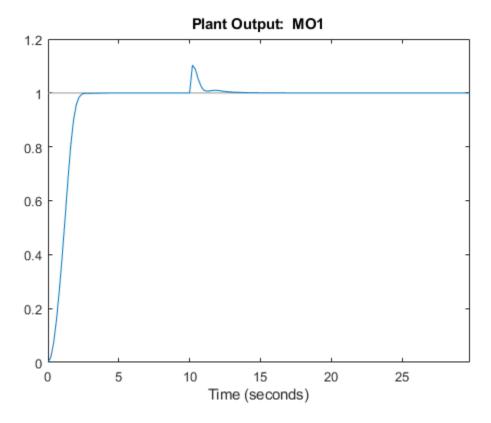
MV constraint has been slightly violated while MO constraint has been violated more. Penalize more output constraint and rerun the simulation.

```
mpcobj.OV.MaxECR = 0.001;   % The closer to zero, the harder the constraint
sim(mpcobj,Tf,r,v)
```

```
   Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

Plant Output: MO1

Now MO constraint has been slightly violated while MV constraint has been violated more as expected.

Change Kalman Gains Used in the Built-In State Estimator

Model Predictive Control Toolbox™ software provides a default Kalman filter to estimate the state of plant, disturbance, and noise models. You can change the Kalman gains.

First, retrieve the default Kalman gains and state-space matrices.

```
[L,M,A1,Cm1] = getEstimator(mpcobj);
```

The default observer poles are:

```
e = eig(A1-A1*M*Cm1)
```

3-73

```
e =

   0.5708 + 0.4144i
   0.5708 - 0.4144i
   0.4967 + 0.0000i
   0.9334 + 0.1831i
   0.9334 - 0.1831i
   0.8189 + 0.0000i
```

Design a new state estimator by pole-placement.

```
poles = [.8 .75 .7 .85 .6 .81];
L = place(A1',Cm1',poles)';
M = A1\L;
setEstimator(mpcobj,L,M);
```

Simulate Open-Loop Response
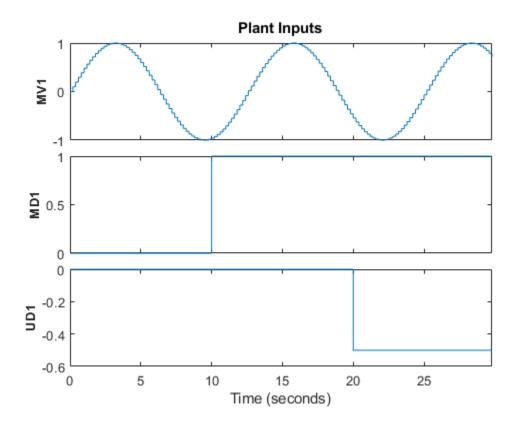
Test the behavior of plant in open-loop using the `sim` command. Set the `OpenLoop` flag to `on`, and provide the sequence of manipulated variables that excite the system.
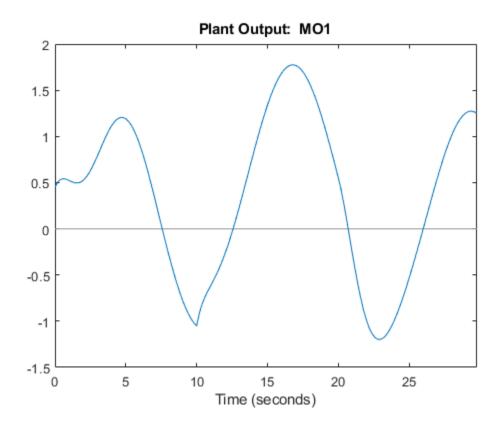
```
SimOptions.OpenLoop = 'on';
SimOptions.MVSignal = sin((0:Tf-1)'/10);
```

As the reference signal will be ignored, we can avoid specifying it.

```
sim(mpcobj,Tf,[],v,SimOptions)
```

```
   Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
-->Converting model to discrete time.
```

Plant Output: MO1

Examine Steady-State Offset

To examine whether the MPC controller will be able to reject constant output disturbances and track constant setpoint with zero offsets in steady-state, compute the DC gain from output disturbances to controlled outputs using the `cloffset` command.

```
DC = cloffset(mpcobj);
fprintf('DC gain from output disturbance to output = %5.8f (=%g) \n',DC,DC);
```

```
DC gain from output disturbance to output = 0.00000000 (=3.55271e-15)
```
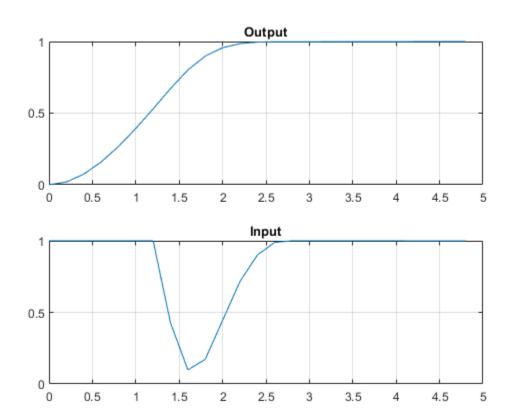
A zero gain means that the output will track the desired setpoint.

Simulate Controller Using `mpcmove`

First, obtain the discrete-time state-space matrices of the plant.
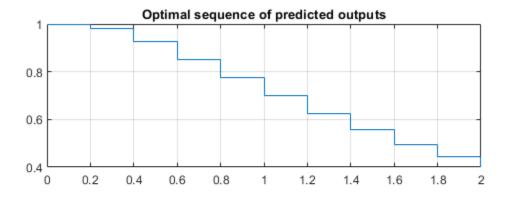
```
[A,B,C,D] = ssdata(model);
Tstop = 5;                  % Simulation time
x = [0 0 0 0 0]';           % Initial state of the plant
xmpc = mpcstate(mpcobj);    % Initial state of the MPC controller
r = 1;                      % Output reference signal
```

Store the closed-loop MPC trajectories in arrays YY, UU, and XX.

```
YY=[];
UU=[];
XX=[];
```

Run the simulation loop

```
for t=0:round(Tstop/Ts)-1
    % Store states
    XX = [XX,x]; %#ok<*AGROW>
    % Define measured disturbance signal
    v = 0;
    if t*Ts>=10
        v = 1;
    end
    % Define unmeasured disturbance signal
    d = 0;
    if t*Ts>=20
        d = -0.5;
    end
    % Plant equations: output update (no feedthrough from MV to Y)
    y = C*x + D(:,2)*v + D(:,3)*d;
    YY = [YY,y];
    % Compute MPC action
    u = mpcmove(mpcobj,xmpc,y,r,v);
    % Plant equations: state update
    x = A*x + B(:,1)*u + B(:,2)*v + B(:,3)*d;
    UU = [UU,u];
end
```

Plot the results.

```
figure
subplot(2,1,1)
```
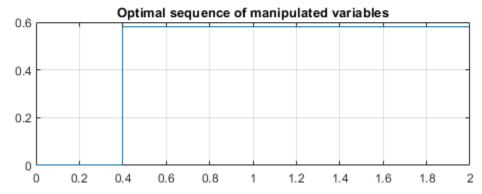
```
plot(0:Ts:Tstop-Ts,YY)
grid
title('Output')
subplot(2,1,2)
plot(0:Ts:Tstop-Ts,UU)
grid
title('Input')
```


If at any time during the simulation you want to check the optimal predicted trajectories, it can be returned by mpcmove as well. Assume you want to start from the current state and have a set-point change to 0.5, and assume the measured disturbance is gone.

```
r = 0.5;
v = 0;
[~,Info] = mpcmove(mpcobj,xmpc,y,r,v);
```

Extract the optimal predicted trajectories.

```
topt = Info.Topt;
yopt = Info.Yopt;
uopt = Info.Uopt;
figure
subplot(2,1,1)
stairs(topt,yopt)
title('Optimal sequence of predicted outputs')
grid
subplot(2,1,2)
stairs(topt,uopt)
title('Optimal sequence of manipulated variables')
grid
```

Linear Representation of MPC Controller

When the constraints are not active, the MPC controller behaves like a linear controller. You can get the state-space form of the MPC controller.

```
LTI = ss(mpcobj,'rv');
```

Get the state-space matrices for the linearized controller.

```
[AL,BL,CL,DL] = ssdata(LTI);
```

Simulate linear MPC closed-loop system and compare the linearized MPC controller with the original MPC controller with constraints turned off.

```matlab
mpcobj.MV = [];              % No input constraints
mpcobj.OV = [];              % No output constraints
Tstop = 5;                   %Simulation time
xL = zeros(size(BL,1),1);    % Initial state of linearized MPC controller
x = [0 0 0 0 0]';            % Initial state of plant
y = 0;                       % Initial measured output
r = 1;                       % Output reference set-point
u = 0;                       % Previous input command
YY = [];
XX = [];
xmpc = mpcstate(mpcobj);
for t = 0:round(Tstop/Ts)-1
    YY = [YY,y];
    XX = [XX,x];
    v = 0;
    if t*Ts>=10
        v = 1;
    end
    d = 0;
    if t*Ts>=20
        d = -0.5;
    end
    uold = u;
    % Compute the linear MPC control action
    u = CL*xL+DL*[y;r;v];
    % Compare the input move with the one provided by MPCMOVE
    uMPC = mpcmove(mpcobj,xmpc,y,r,v);
    dispStr(t+1) = {sprintf('t=%5.2f, u=%7.4f (provided by LTI), u=%7.4f (provided by M
    % Update plant equations
    x = A*x + B(:,1)*u + B(:,2)*v + B(:,3)*d;
    % Update controller equations
    xL = AL*xL + BL*[y;r;v];
    % Update output equations
    y = C*x + D(:,1)*u + D(:,2)*v + D(:,3)*d;
end
```

```
    Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

Display the results.

```matlab
for t=0:round(Tstop/Ts)-1
    disp(dispStr{t+1});
end
```

```
t= 0.00, u= 5.2478 (provided by LTI), u= 5.2478 (provided by MPCOBJ)
t= 0.20, u= 3.0134 (provided by LTI), u= 3.0134 (provided by MPCOBJ)
t= 0.40, u= 0.2281 (provided by LTI), u= 0.2281 (provided by MPCOBJ)
t= 0.60, u=-0.9952 (provided by LTI), u=-0.9952 (provided by MPCOBJ)
t= 0.80, u=-0.8749 (provided by LTI), u=-0.8749 (provided by MPCOBJ)
t= 1.00, u=-0.2022 (provided by LTI), u=-0.2022 (provided by MPCOBJ)
t= 1.20, u= 0.4459 (provided by LTI), u= 0.4459 (provided by MPCOBJ)
t= 1.40, u= 0.8489 (provided by LTI), u= 0.8489 (provided by MPCOBJ)
t= 1.60, u= 1.0192 (provided by LTI), u= 1.0192 (provided by MPCOBJ)
t= 1.80, u= 1.0511 (provided by LTI), u= 1.0511 (provided by MPCOBJ)
t= 2.00, u= 1.0304 (provided by LTI), u= 1.0304 (provided by MPCOBJ)
t= 2.20, u= 1.0053 (provided by LTI), u= 1.0053 (provided by MPCOBJ)
t= 2.40, u= 0.9920 (provided by LTI), u= 0.9920 (provided by MPCOBJ)
t= 2.60, u= 0.9896 (provided by LTI), u= 0.9896 (provided by MPCOBJ)
t= 2.80, u= 0.9925 (provided by LTI), u= 0.9925 (provided by MPCOBJ)
t= 3.00, u= 0.9964 (provided by LTI), u= 0.9964 (provided by MPCOBJ)
t= 3.20, u= 0.9990 (provided by LTI), u= 0.9990 (provided by MPCOBJ)
t= 3.40, u= 1.0002 (provided by LTI), u= 1.0002 (provided by MPCOBJ)
t= 3.60, u= 1.0004 (provided by LTI), u= 1.0004 (provided by MPCOBJ)
t= 3.80, u= 1.0003 (provided by LTI), u= 1.0003 (provided by MPCOBJ)
t= 4.00, u= 1.0001 (provided by LTI), u= 1.0001 (provided by MPCOBJ)
t= 4.20, u= 1.0000 (provided by LTI), u= 1.0000 (provided by MPCOBJ)
t= 4.40, u= 0.9999 (provided by LTI), u= 0.9999 (provided by MPCOBJ)
t= 4.60, u= 1.0000 (provided by LTI), u= 1.0000 (provided by MPCOBJ)
t= 4.80, u= 1.0000 (provided by LTI), u= 1.0000 (provided by MPCOBJ)
```
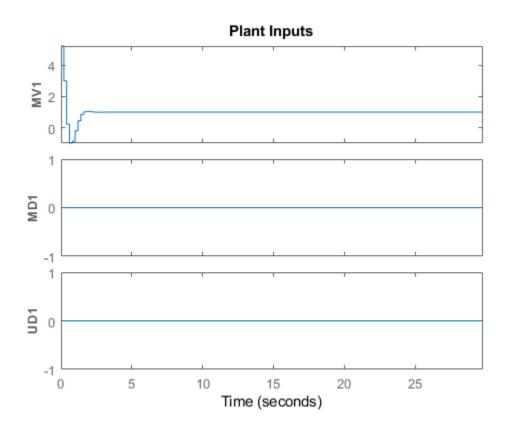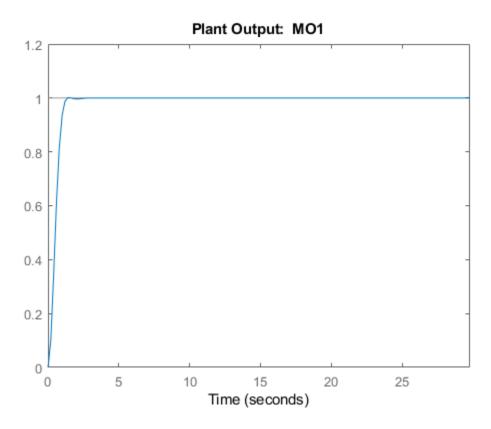
Plot the results.

```
figure
plot(0:Ts:Tstop-Ts,YY)
grid
```

Running a closed-loop where all constraints are turned off is easy using `sim`. We just specify an option in the `SimOptions` structure:

```
SimOptions = mpcsimopt(mpcobj);
SimOptions.Constr = 'off';      % Remove all MPC constraints
SimOptions.Unmeas = d;          % unmeasured input disturbance
sim(mpcobj,Tf,r,v,SimOptions);
```

Plant Output: MO1

Simulate Using Simulink®

To run this example, Simulink® is required.

```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this part of the example.')
    return
end
```

Recreate the MPC controller.

```
mpcobj = mpc(model,Ts,10,3);
mpcobj.MV = struct('Min',0,'Max',1,'RateMin',-10,'RateMax',10);
mpcobj.Model.Disturbance = tf(sqrt(1000),[1 0]);
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1.
```
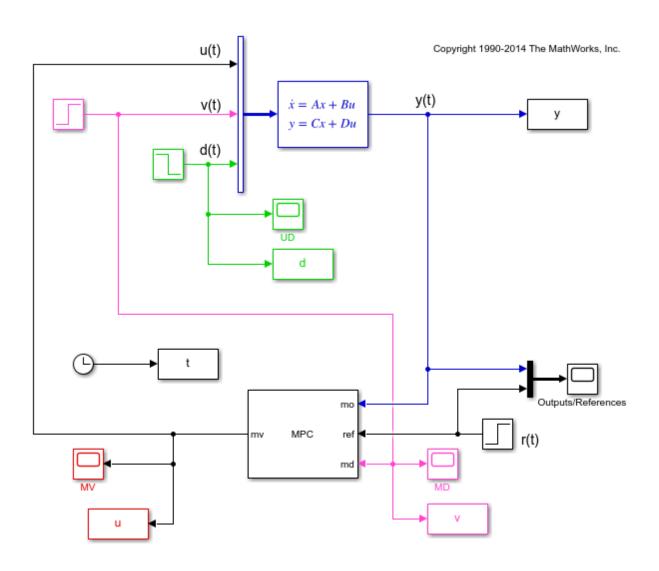
The continuous-time plant to be controlled has the following state-space realization:
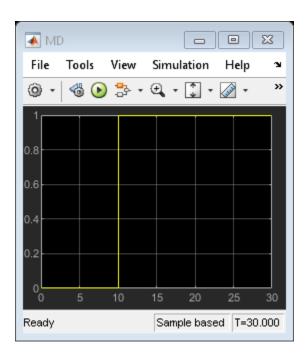
```
[A,B,C,D] = ssdata(sys);
```
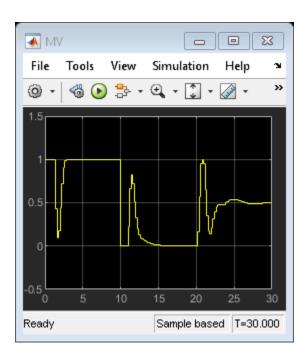
Simulate closed-loop MPC in Simulink.
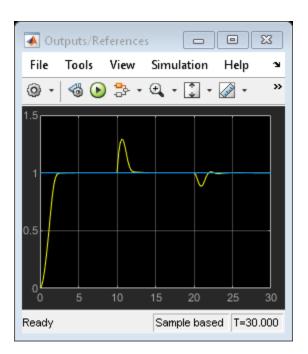
```
mdl1 = 'mpc_miso';
open_system(mdl1)
sim(mdl1)
```
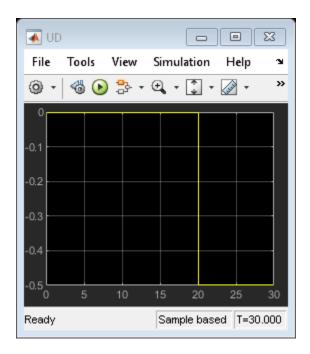
```
   Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

u(t)

Copyright 1990-2014 The MathWorks, Inc.

v(t)

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

y(t)

y

d(t)

UD

d

t

mo

mv   MPC   ref

md

MV

Outputs/References

r(t)

MD

u

v

Next, run a simulation with sinusoidal output noise. Assume output measurements are affected by a sinusoidal measurement noise of frequency 0.1 Hz. You want to inform the MPC object about this so that state estimates can be improved.

```
omega = 2*pi/10;
mpcobj.Model.Noise = 0.5*tf(omega^2,[1 0 omega^2]);
```
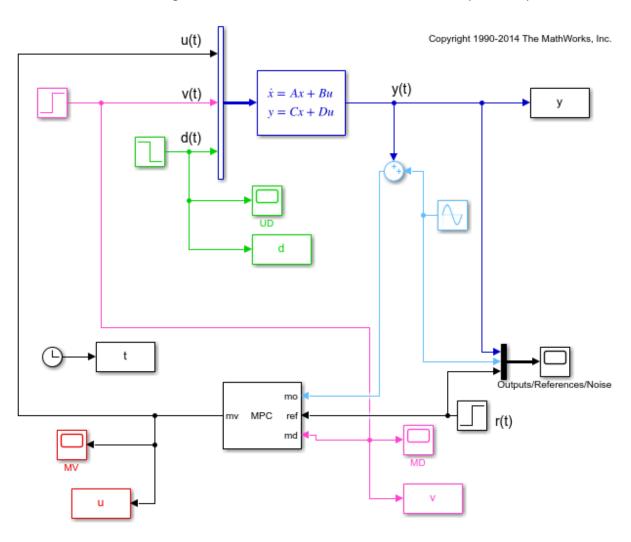
Revise the MPC design and specify a white Gaussian noise unmeasured disturbance with zero mean and variance `0.1`.
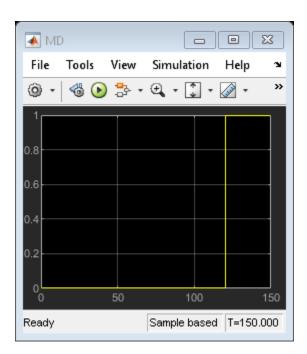
```
mpcobj.Model.Disturbance = tf(0.1);
mpcobj.weights = struct('MV',0,'MVRate',0.1,'OV',0.005);
mpcobj.predictionhorizon = 40;
mpcobj.controlhorizon = 3;
```
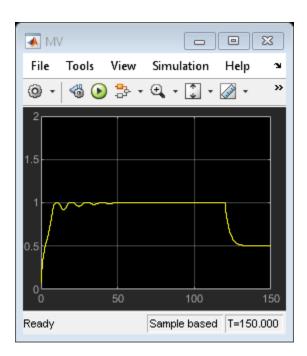
Run the simulation.
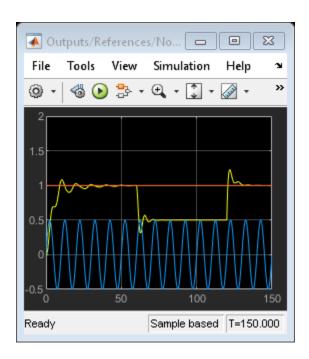
```
Tstop = 150;
mdl2 = 'mpc_misonoise';
open_system(mdl2)
sim(mdl2,Tstop)
```
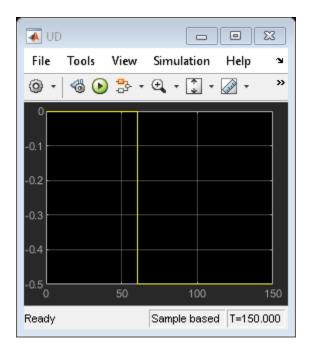
3-91

-->Assuming output disturbance added to measured output channel #1 is integrated white
-->A feedthrough channel in NoiseModel was inserted to prevent problems with estimator

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Copyright 1990-2014 The MathWorks, Inc.

```
bdclose(mdl1)
bdclose(mdl2)
```

See Also

MPC Controller | **MPC Designer** | mpc

More About

- "Control of a Single-Input-Single-Output Plant" on page 3-60
- "Control of a Multi-Input Multi-Output Nonlinear Plant" on page 3-97

Control of a Multi-Input Multi-Output Nonlinear Plant

This example shows how to design a model predictive controller for a multi-input multi-output nonlinear plant. The plant has 3 manipulated variables and 2 measured outputs.

Linearize Nonlinear Plant

To run this example, Simulink® and Simulink Control Design™ are required.

```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
if ~mpcchecktoolboxinstalled('slcontrol')
    disp('Simulink Control Design(R) is required to run this example.')
    return
end
```

The nonlinear plant is implemented in Simulink model mpc_nonlinmodel and linearized at the default operating condition using the linearize command from Simulink Control Design.

```
plant = linearize('mpc_nonlinmodel');
```

Assign names to I/O variables.

```
plant.InputName = {'Mass Flow';'Heat Flow';'Pressure'};
plant.OutputName = {'Temperature';'Level'};
plant.InputUnit = {'kg/s' 'J/s' 'Pa'};
plant.OutputUnit = {'K' 'm'};
```

Design MPC Controller

Create the controller object with sampling period, prediction and control horizons:

```
Ts = 0.2;
p = 5;
m = 2;
mpcobj = mpc(plant,Ts,p,m);
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
```

Specify MV constraints.

```
mpcobj.MV = struct('Min',{-3;-2;-2},'Max',{3;2;2},'RateMin',{-1000;-1000;-1000});
```
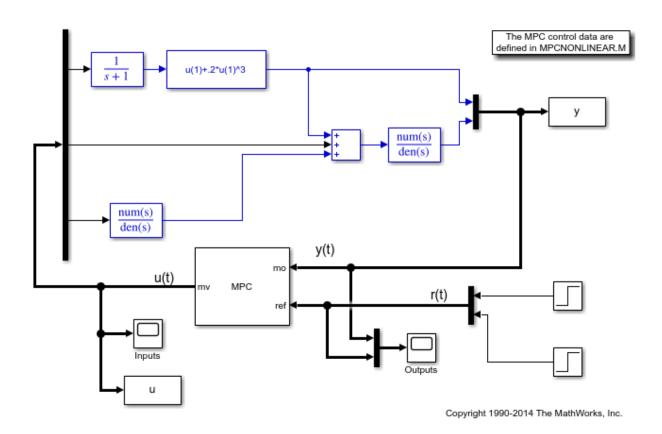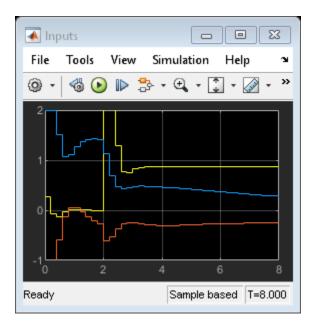
Define weights on manipulated and controlled variables.

```
mpcobj.Weights = struct('MV',[0 0 0],'MVRate',[.1 .1 .1],'OV',[1 1]);
```
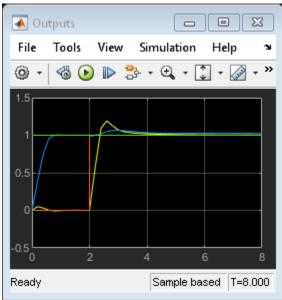
Simulate Using Simulink

Run simulation.

```
mdl1 = 'mpc_nonlinear';
open_system(mdl1)
sim(mdl1)

-->Converting model to discrete time.
-->Assuming output disturbance added to measured output channel #1 is integrated white
-->Assuming output disturbance added to measured output channel #2 is integrated white
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

The MPC control data are defined in MPCNONLINEAR.M

$$\frac{1}{s+1}$$

u(1)+.2*u(1)^3

$$\frac{num(s)}{den(s)}$$

$$\frac{num(s)}{den(s)}$$

y

y(t)

mo

MPC

u(t)

mv

ref

r(t)

Inputs

u

Outputs

Copyright 1990-2014 The MathWorks, Inc.
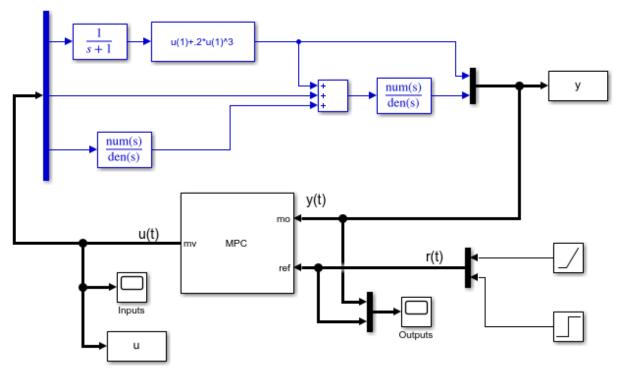
Modify MPC Design to Track Ramp Signals

In order to track a ramp, a triple integrator is defined as an output disturbance model on both outputs.

```
outdistmodel = tf({1 0;0 1},{[1 0 0 0],1;1,[1 0 0 0]});
setoutdist(mpcobj,'model',outdistmodel);
```

Run simulation.
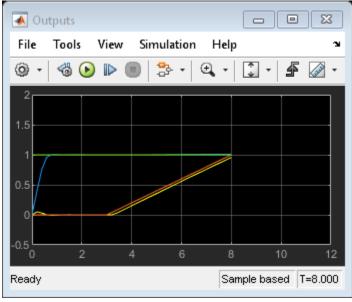
```
mdl2 = 'mpc_nonlinear_setoutdist';
open_system(mdl2)
sim(mdl2)
```

```
-->Converting model to discrete time.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```


Copyright 1990-2014 The MathWorks, Inc.

Simulate without Constraints

When the constraints are not active, the MPC controller behaves like a linear controller.

```
mpcobj.MV = [];
```

Reset output disturbance model to default
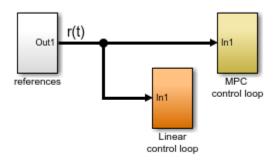
```
setoutdist(mpcobj,'integrators');
```

The input to the linear controller LTI is the vector [ym;r], where ym is the vector of measured outputs, and r is the vector of output references.
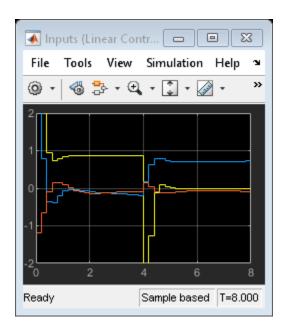
```
LTI = ss(mpcobj,'r');
```
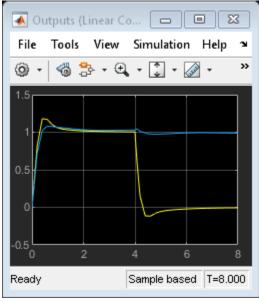
```
-->Converting model to discrete time.
-->Assuming output disturbance added to measured output channel #1 is integrated white
-->Assuming output disturbance added to measured output channel #2 is integrated white
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```
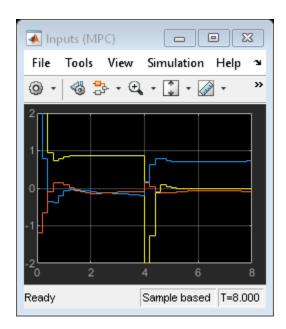
Run simulation.

```
refs = [1;1];                    % output references are step signals
mdl3 = 'mpc_nonlinear_ss';
open_system(mdl3)
sim(mdl3)
```
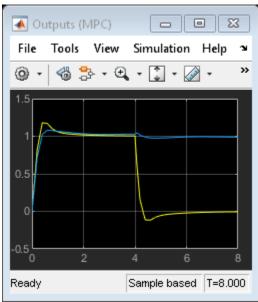

Copyright 1990-2014 The MathWorks, Inc.

Compare Simulation Results

```
fprintf('Compare output trajectories: ||ympc-ylin|| = %g\n',norm(ympc-ylin));
disp('The MPC controller and the linear controller produce the same closed-loop traject
```

```
Compare output trajectories: ||ympc-ylin|| = 1.50163e-14
The MPC controller and the linear controller produce the same closed-loop trajectories
```

```
bdclose(mdl1)
bdclose(mdl2)
bdclose(mdl3)
```

See Also

MPC Controller | **MPC Designer** | mpc

More About

- "Control of a Single-Input-Single-Output Plant" on page 3-60
- "Control of a Multi-Input Single-Output Plant" on page 3-63