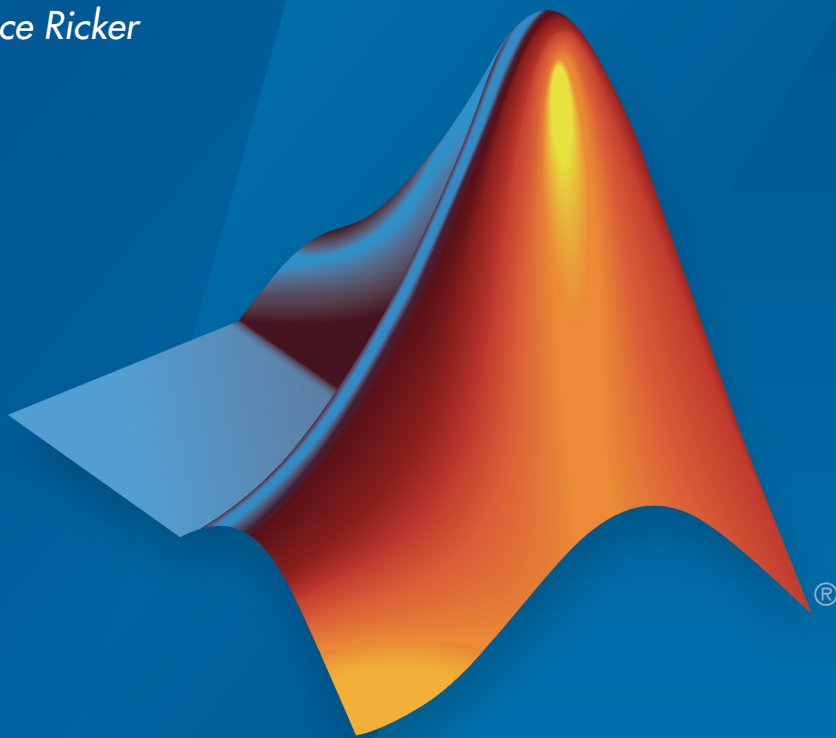# Model Predictive Control Toolbox™

Getting Started Guide

Alberto Bemporad
Manfred Morari
N. Lawrence Ricker

MATLAB®

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.
3 Apple Hill Drive
Natick, MA 01760-2098

Contents

Designing Controllers Using MPC Designer

3

Designing Controllers Using the Command Line

4

Designing and Testing Controllers in Simulink

5

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 systematically analyzing, designing, and simulating model predictive controllers. You can specify plant and disturbance models, horizons, constraints, and weights. The toolbox enables you to diagnose issues that could lead to run-time failures and provides advice on tuning weights to improve performance and robustness. By running different scenarios in linear and nonlinear simulations, you can evaluate controller performance.

You can adjust controller performance as it runs by tuning weights and varying constraints. You can implement adaptive model predictive controllers by updating the plant model at run time. For applications with fast sample times, you can develop explicit model predictive controllers. For rapid prototyping and embedded system design, the toolbox supports C-code and IEC 61131-3 Structured Text generation.

Key Features

- Design and simulation of model predictive controllers in MATLAB® and Simulink
- Customization of constraints and weights with advisory tools for improved performance and robustness
- Adaptive MPC control through run-time changes to internal plant model
- Explicit MPC control for applications with fast sample times using pre-computed solutions
- Control of plants over a wide range of operating conditions by switching between multiple model predictive controllers
- Specialized model predictive control quadratic programming (QP) solver optimized for speed, efficiency, and robustness
- 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.

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.

N. Lawrence Ricker

Professor of Chemical Engineering, University of Washington, Seattle, USA. Research interests include model predictive control and process optimization. Author of the quadratic programming solver and graphical user interface.

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

In this section...
"Plant Model" on page 2-2
"Input Disturbance Model" on page 2-4
"Output Disturbance Model" on page 2-5
"Measurement Noise Model" on page 2-6

The model structure used in the MPC controller appears in the following illustration. This section explains how the models connect for prediction and state estimation.
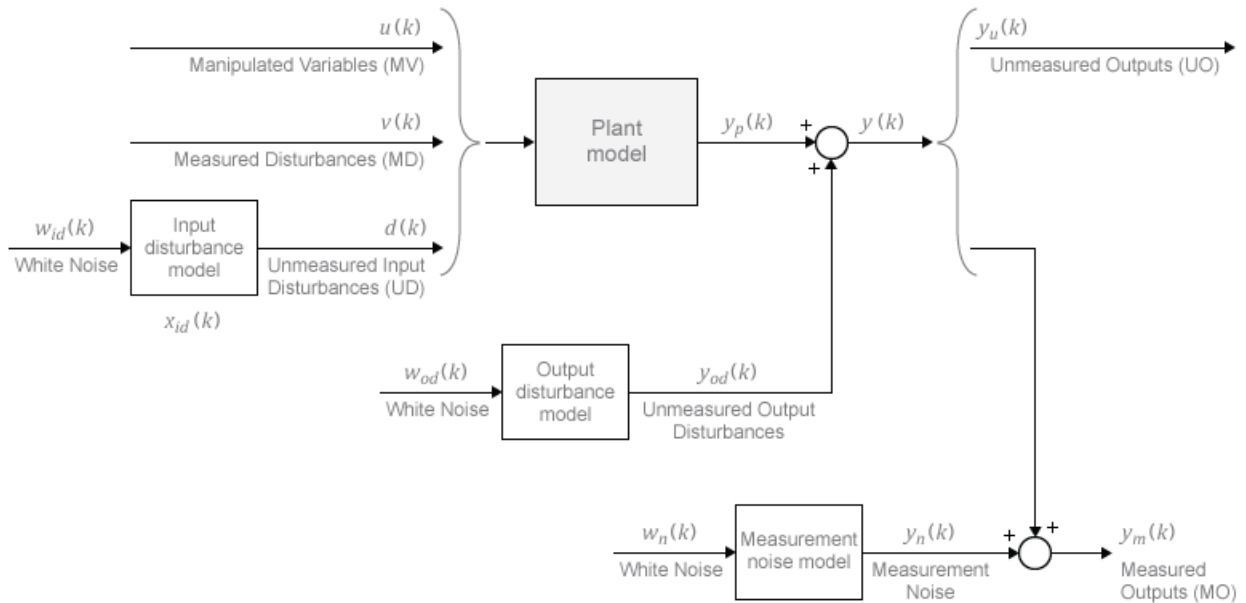
Plant Model

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:

$$x_p(k+1) = A_p x_p(k) + BS_i u_p(k)$$
$$y_p(k) = S_o^{-1} C x_p(k) + S_o^{-1} DS_i u_p(k).$$

where A_p, B, C, and D are the constant state-space matrices determined in 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. No scaling is performed on state variables.

2-3

- u_p is a vector of dimensionless plant input variables.
- 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. That is, 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. The result is:

$$
\begin{aligned}
x_{id}(k+1) &= A_{id}x_{id}(k) + B_{id}w_{id}(k) \\
d(k) &= C_{id}x_{id}(k) + D_{id}w_{id}(k).
\end{aligned}
$$

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 unity 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, the MPC controller converts the specified output disturbance model to a discrete-time, delay-free, LTI state-space system. The result is:

$$
\begin{aligned}
x_{od}(k+1) &= A_{od}x_{od}(k) + B_{od}w_{od}(k) \\
y_{od}(k) &= C_{od}x_{od}(k) + D_{od}w_{od}(k).
\end{aligned}
$$

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, 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 $1\text{x}10^{-8}$, the MPC controller adds $1\text{x}10^{-4}$ to each diagonal element of D_n. This adjustment makes a successful default Kalman gain calculation more likely.

More About

- "Controller State Estimation"

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 2-13 for more details.

More About

- "MPC Modeling" on page 2-2

Construct Linear Time Invariant (LTI) 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. It's also easy to convert from one format to another. For more details, see the Control System Toolbox documentation.

In this section...
"Transfer Function Models" on page 2-9
"Zero/Pole/Gain Models" on page 2-9
"State-Space Models" on page 2-10
"LTI Object Properties" on page 2-12
"LTI Model Characteristics" on page 2-15

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)
```

Control System Toolbox software builds and displays it as follows:

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

Zero/Pole/Gain Models

Like the TF, 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. The primes (e.g., C_A') denote a deviation from the nominal steady-state condition at which the model has been linearized.
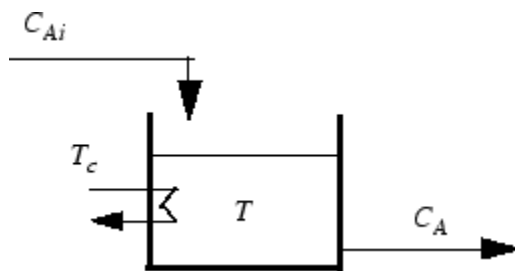
CSTR Schematic

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 2-9 and "Zero/Pole/Gain Models" on page 2-9 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)

Plant Inputs	Plant Outputs
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 the MPC Designer app.

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.
`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

Related Examples

- "Specify Multi-Input Multi-Output (MIMO) Plants" on page 2-17

Specify Multi-Input Multi-Output (MIMO) Plants

This example shows hot to specify plants having multiple inputs and outputs.

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 (LTI) Models" on page 2-9

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 must 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 the MPC Designer app.

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 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 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:      -8.88e-15 (0)
(2.) CSTR_OpenLoop/CSTR/T_K
      x:              377     dx:       1.14e-13 (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/Feed Concentration', 1, 'input');
io(2) = linio('CSTR_OpenLoop/Feed Temperature',   1, 'input');
io(3) = linio('CSTR_OpenLoop/Coolant 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 =
         Feed Concent  Feed Tempera  Coolant Temp
    C_A            1             0             0
    T_K            0             1           0.3

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

  D =
           Feed Concent  Feed Tempera  Coolant Temp
    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 product.
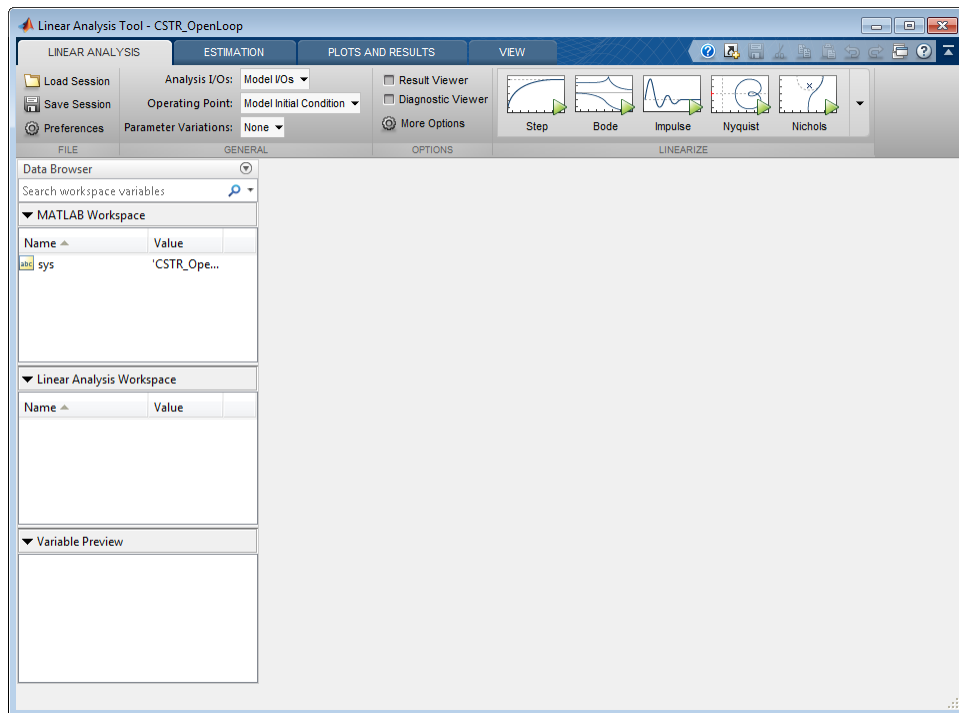
This example uses the CSTR model, `CSTR_OpenLoop`.

Open Simulink Model

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

Open Linear Analysis Tool

In the Simulink model window, select **Analysis** > **Control Design** > **Linear Analysis**.

### 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, right-click the signal in the Simulink model window and select **Linear Analysis Points** > **Input Perturbation**.
- Linearization output, right-click the signal in the Simulink model window and select **Linear Analysis Points** > **Output Measurement**.

Specify Residual Concentration as Known Trim Constraint

In the Simulink model window, right-click the `CA` output signal from the `CSTR` block. Select **Linear Analysis Points** > **Trim Output Constraint**.
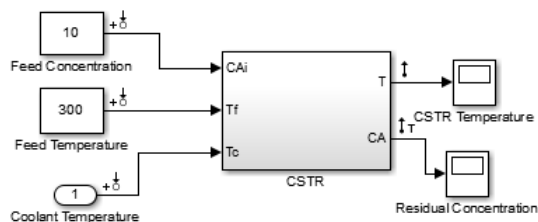
Copyright 1990-2012 The MathWorks, Inc.

In the Linear Analysis Tool, in the **Linear Analysis** tab, in the **Operating Point** drop-down list, select `Trim model`.

In the **Outputs** tab:

- Select the **Known** check box for `Channel - 1` under **CSTR_OpenLoop/CSTR**.
- Set the corresponding **Value** to `2` kmol/m^3.

2-27
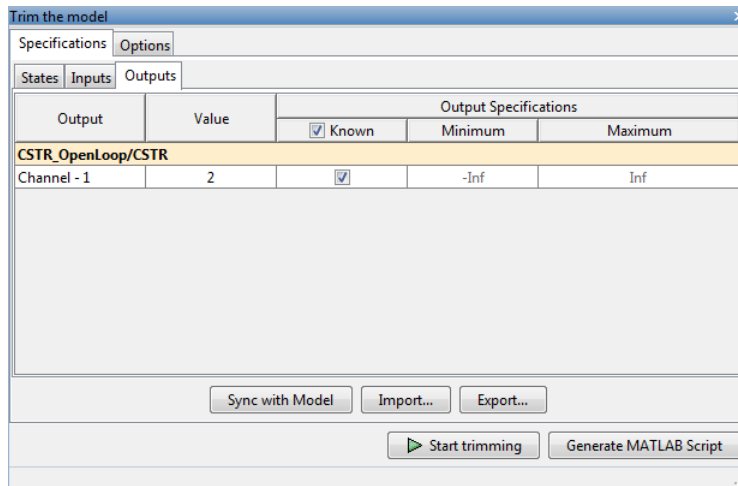
Create and Verify Operating Point

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.

Input	Desired Value	Actual Value
CSTR_OpenLoop/Coolant Temperature		
Input - 1	[-Inf , Inf]	299.0349
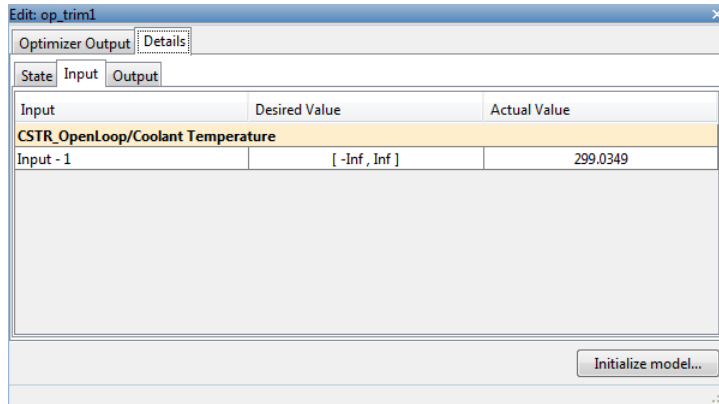
The coolant temperature at steady state is 299 K, as desired.
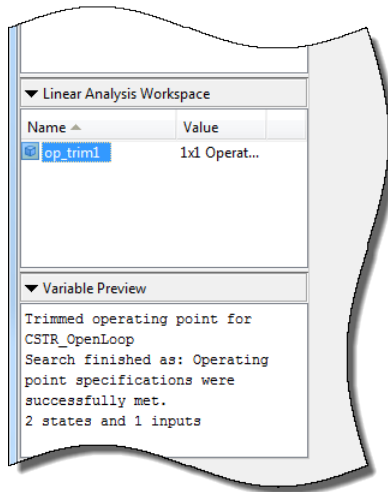
Linearize Model

In the **Linear Analysis** tab, in the **Operating Point** drop-down list, select op_trim1.
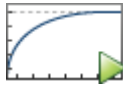
Click **Step** to linearize the model.

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.

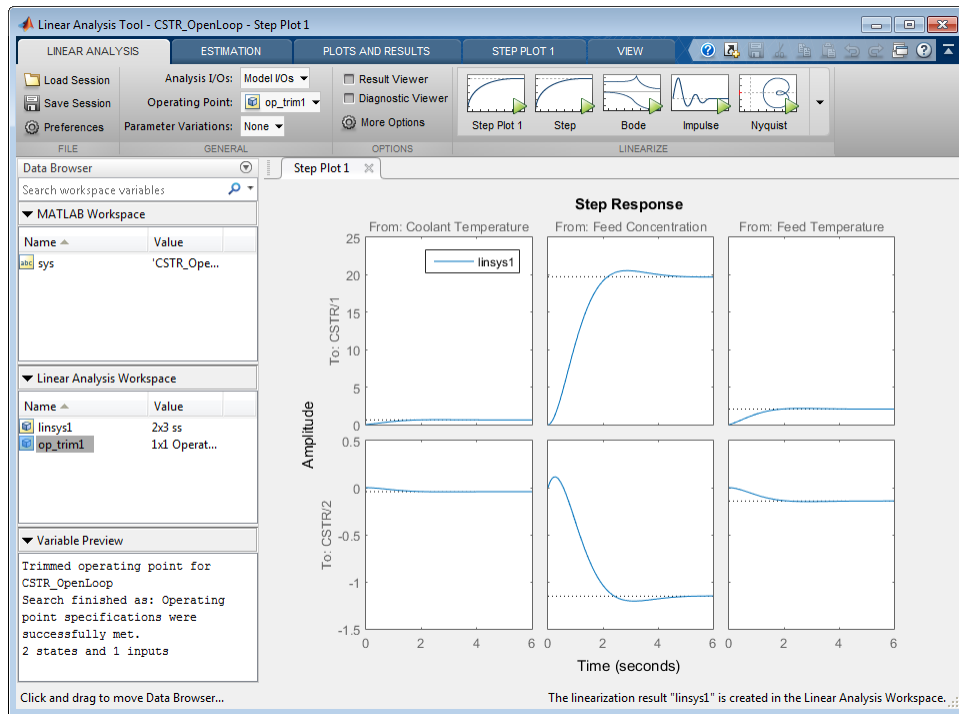The step response from feed concentration to output `CSTR/2` displays an interesting inverse response. An examination of the linear model shows that `CSTR/2` is the residual CSTR concentration, `C_A`. When the feed concentration increases, `C_A` increases initially because more reactant is entering, 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.

Related Examples

· "Design MPC Controller in Simulink" on page 5-2

- "Design Controller Using MPC Designer" on page 3-2
- "Design MPC Controller at the Command Line" on page 4-2

Linearize Simulink Models Using MPC Designer

This topic shows how to linearize Simulink models using the MPC Designer app. To do so, start the app from a Simulink model that contains an `MPC Controller` block.

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.
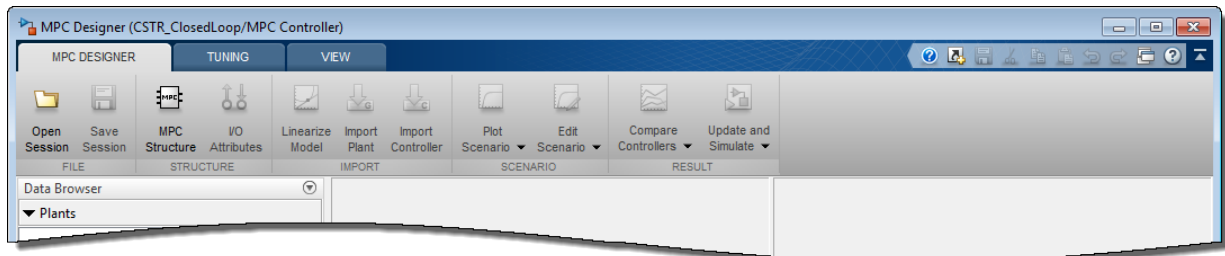
Using the MPC Designer app, you can define the MPC structure by linearizing the Simulink model. After you define the initial MPC structure, you can also linearize the model at different operating points and import the linearized plants.

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 the MPC Designer app by linearizing a Simulink model.

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

Define MPC Structure By Linearization

MPC Structure

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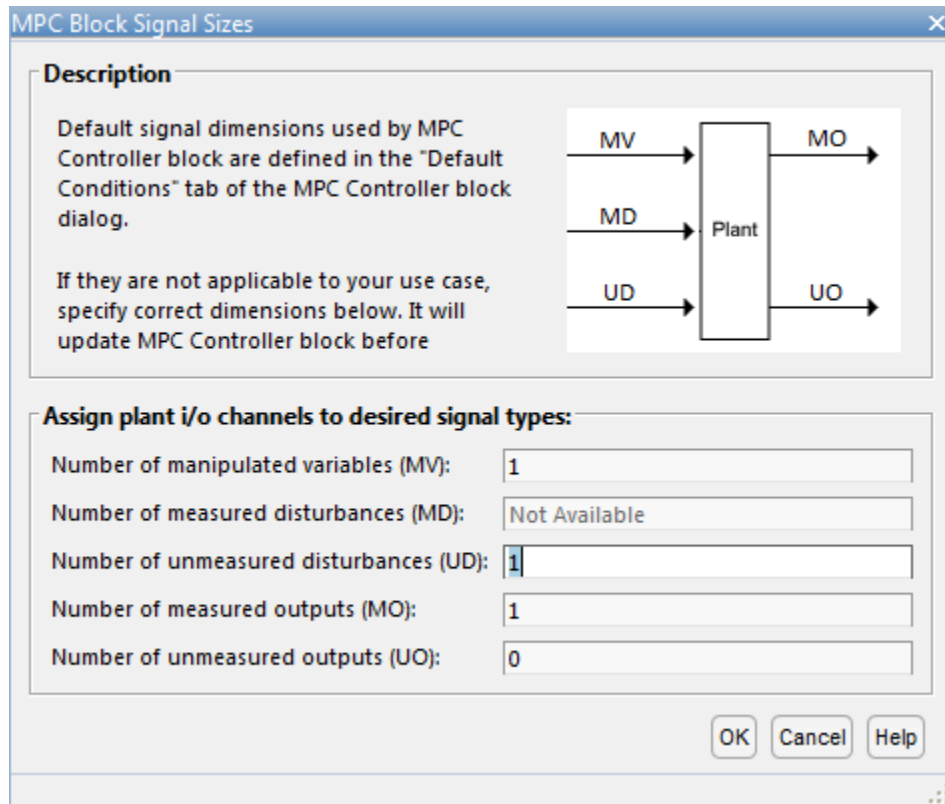
Tip In the MPC Controller Block Parameters dialog box, in the **Default Conditions** tab, you can define the controller sample time and signal dimensions before opening MPC Designer.
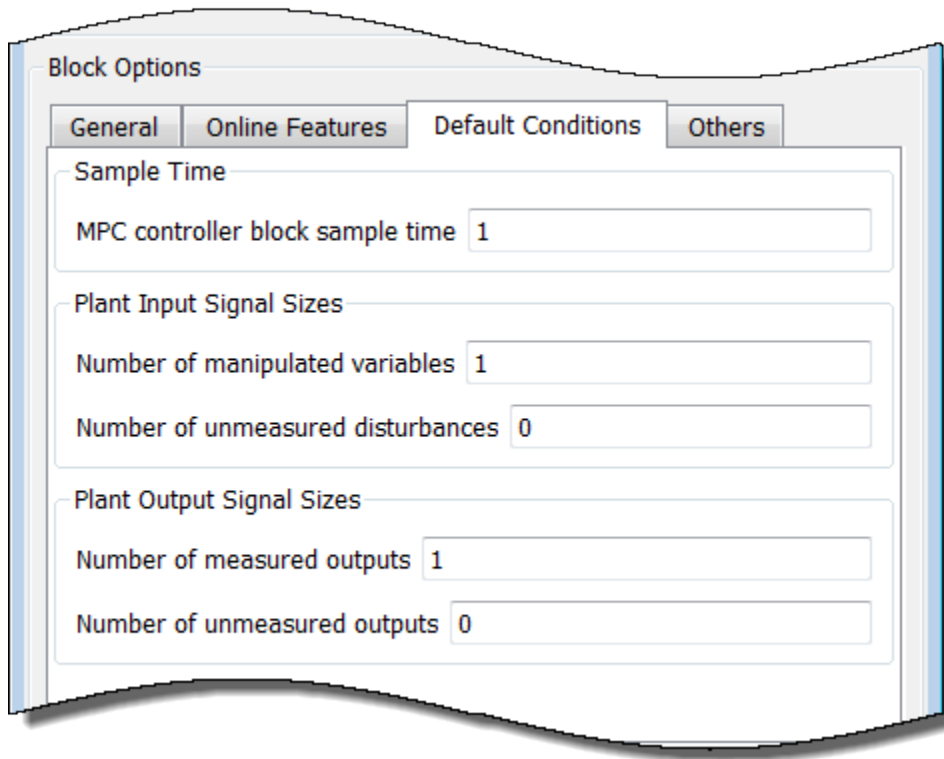
Select Plant Input/Output Signals

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.
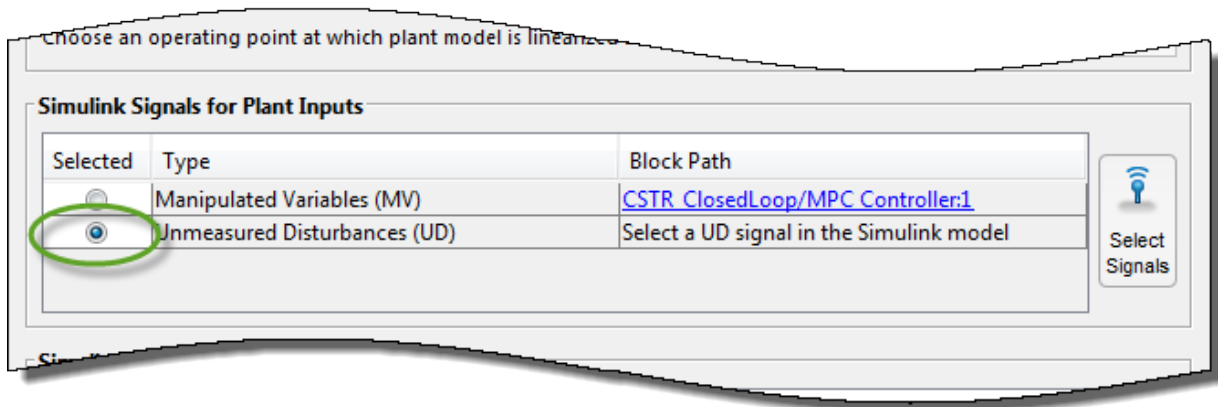
Click **Select Signals**.

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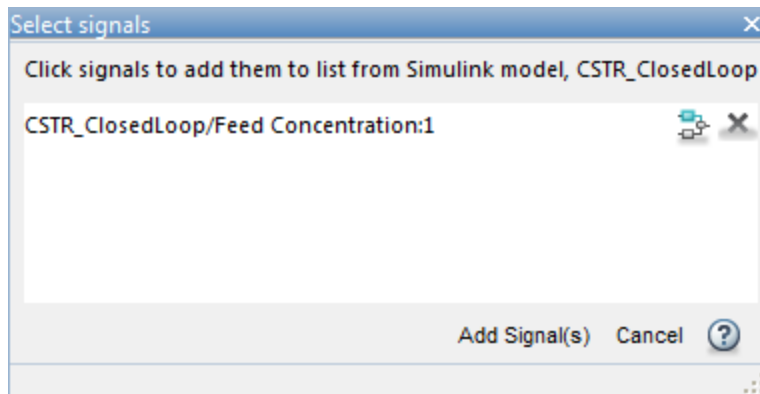
In the Select signals dialog box, click **Add Signal(s)**.

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

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

This example shows how to linearize and import a Simulink model using the MPC Designer app.

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**.

Linearize Simulink Model

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
○	Unmeasured Disturbances (UD)	CSTR_ClosedLoop/Feed Temperature:1	

Simulink Signals for Plant Outputs

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

☐ Use selected operating point to update nominal values as well   [Linearize and Import] [Cancel] [Help]

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" and "Computing Steady-State Operating Points" in the Simulink Control Design documentation.

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**.
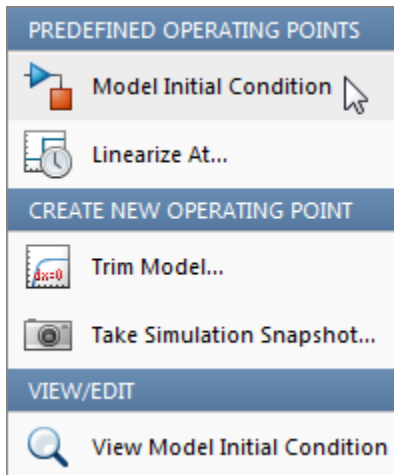
The model initial condition is the default operating point for linearization in the MPC Designer app.

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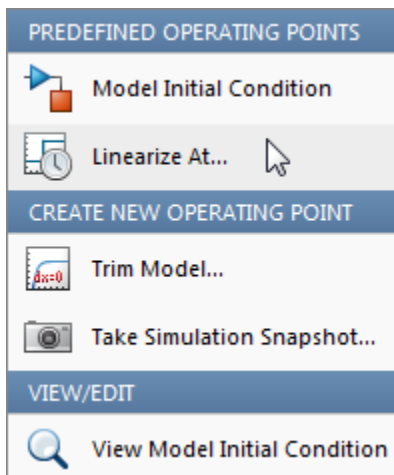
In the Enter snapshot times to linearize dialog box, in the **Simulation snapshot times** field, enter one or more simulation snapshot times. Enter multiple snapshot times as a vector.
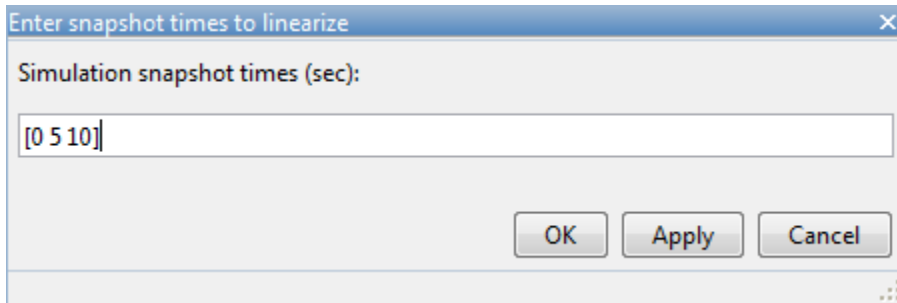

Click **OK**.

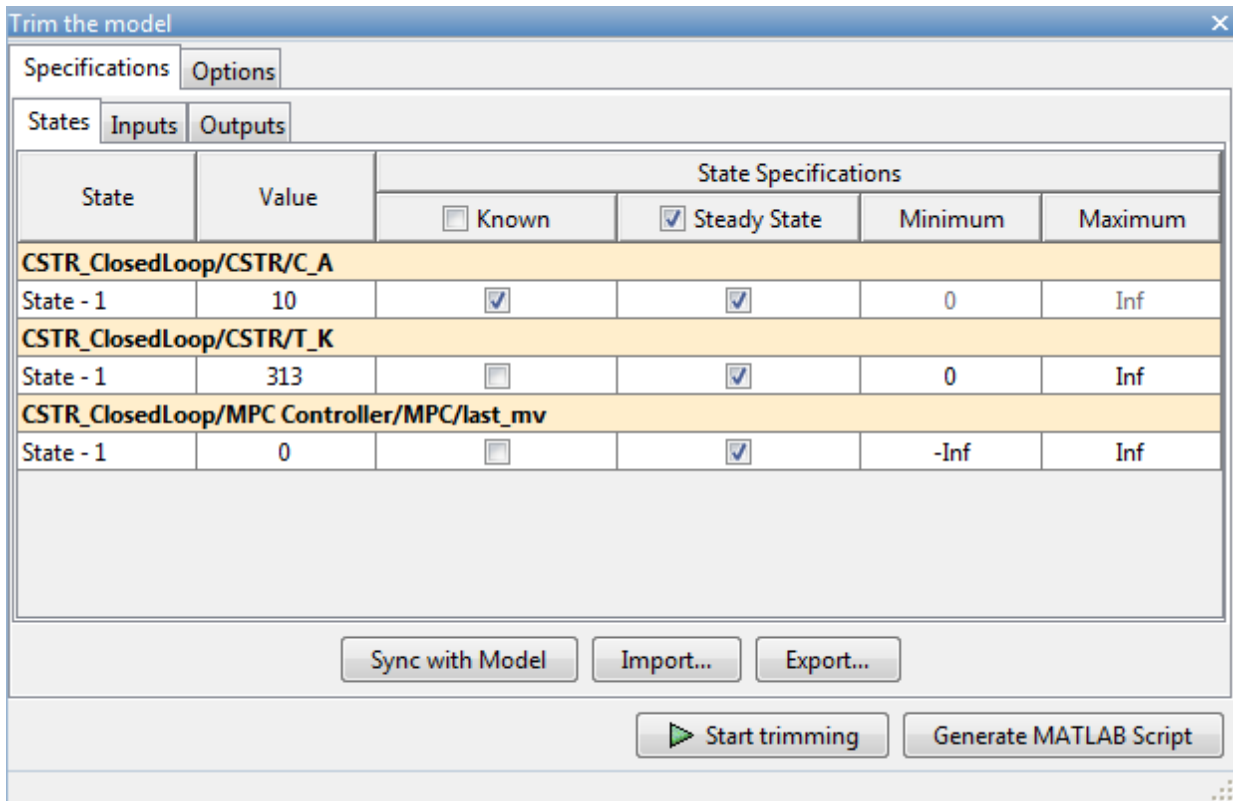If you enter multiple snapshot times, and you selected **Linearize At** from the:

- 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**.
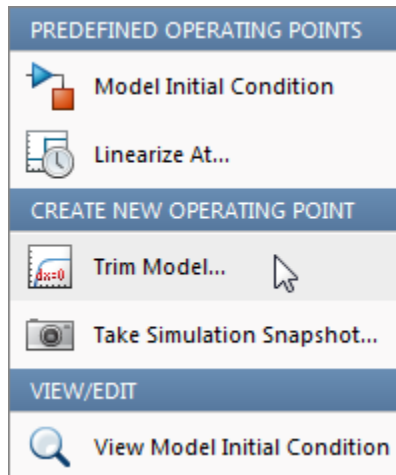
In the Trim the model dialog box, enter the specifications for the steady-state state values at which you want to find an operating point.

| State	Value	State Specifications			
		☐ Known	☑ Steady State	Minimum	Maximum
CSTR_ClosedLoop/CSTR/C_A					
State - 1	10	☑	☑	0	Inf
CSTR_ClosedLoop/CSTR/T_K					
State - 1	313	☐	☑	0	Inf
CSTR_ClosedLoop/MPC Controller/MPC/last_mv					
State - 1	0	☐	☑	-Inf	Inf

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 State Specifications" and "Compute Steady-State Operating Point to Meet Output Specification" in the Simulink Control Design documentation.

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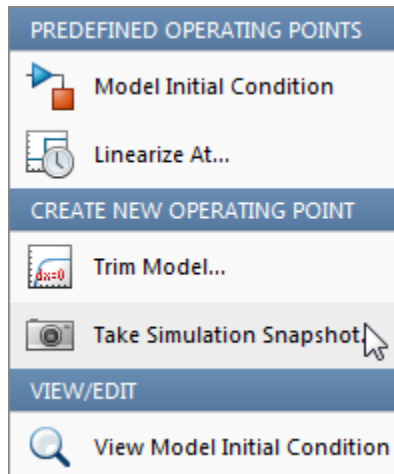
In the Enter snapshot times to linearize dialog box, in the **Simulation snapshot times** field, enter one or more simulation snapshot times. Enter multiple snapshot times as a vector.
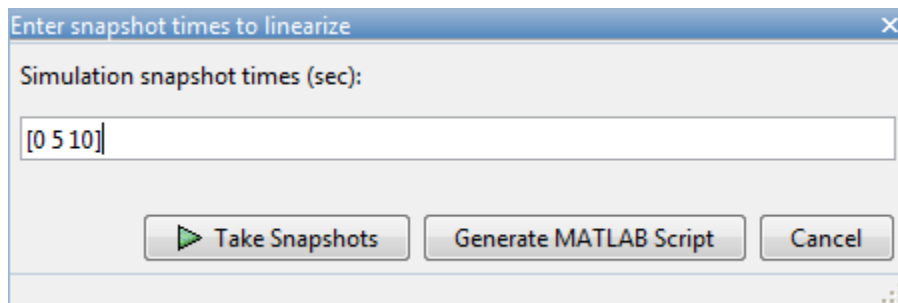
Click **Take Snapshots**.

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 the MPC Designer app, 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.
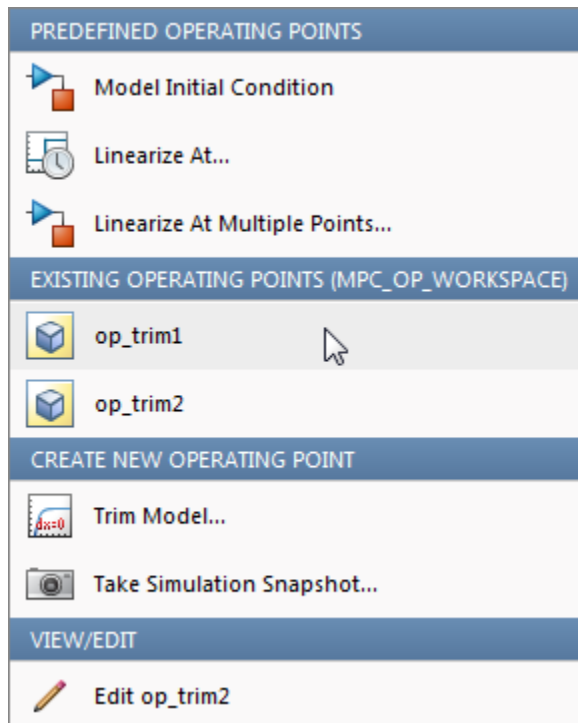
If the selected operating point represents an operating point array created using multiple snapshot times, and you selected an operating point from the:
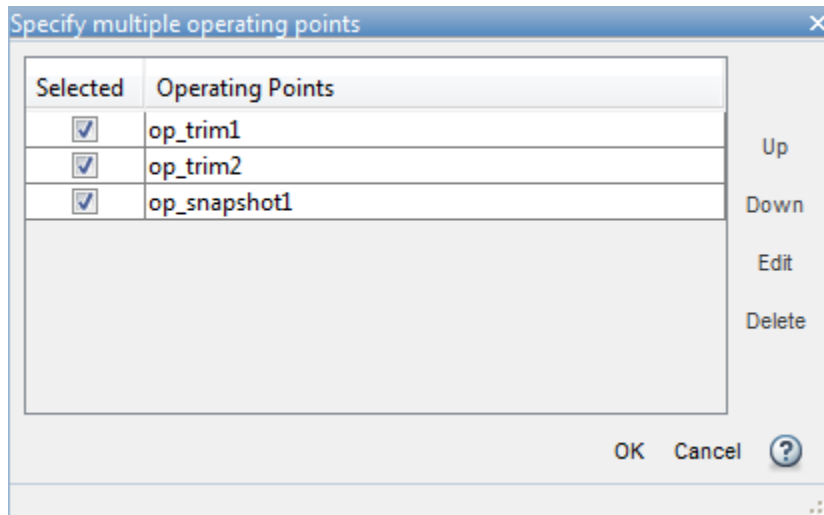
- Define MPC Structure By Linearization dialog box, MPC Designer linearizes the model using only the first operating point in the array. The nominal values of the MPC controller are defined using the input/output signal values for this operating point.
- Linearize Simulink Model dialog box, MPC Designer 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.
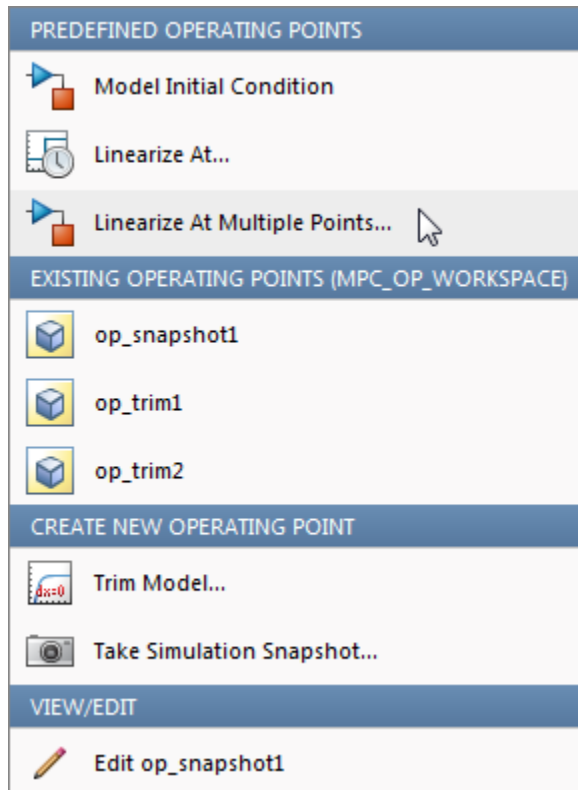
In the Specify multiple operating points dialog box, select the operating points at which to linearize the model.

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.
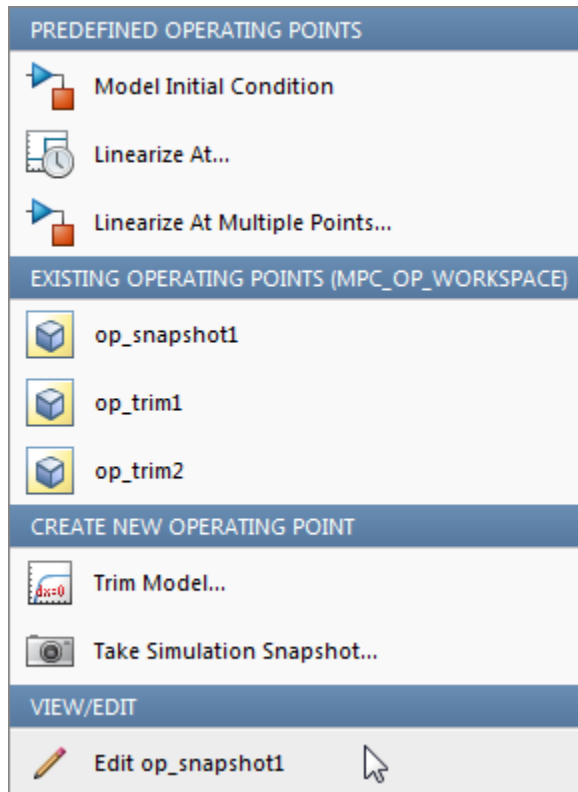
In the Edit dialog box, if you created the selected operating point from a simulation snapshot, you can edit the operating point values.
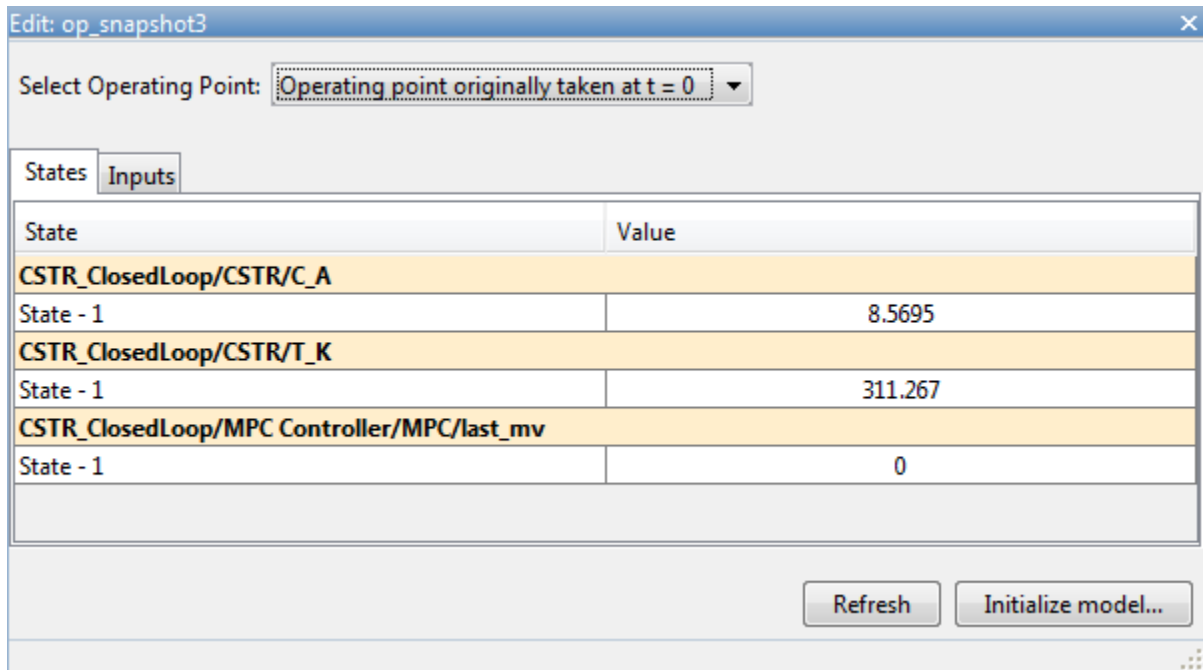
If the selected operating point represents an operating point array, in the **Select Operating Point** drop-down list, select an operating point to view.

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

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	10	10	0	-3.5732e-08
CSTR_ClosedLoop/CSTR/T_K				
State - 1	[0 , Inf]	161.9721	0	4.3283e-07
CSTR_ClosedLoop/MPC Controller/MPC/last_mv				
State - 1	[-Inf , Inf]	-298.1209	0	0

Initialize model...

To simulate the Simulink model at the selected operating point, click **Initialize model** to set the model initial conditions to the states in the operating point.

See Also
MPC Designer

Related Examples
- "Linearize Simulink Models" on page 2-22
- "Design MPC Controller in Simulink" on page 5-2

Identify Plant from Data

This example shows how to identify a linear plant model using measured data.

When you have measured plant input/output data, you can use System Identification Toolbox™ software to estimate a linear plant model. Then, you can use the estimated plant model to create a model predictive controller.

You can estimate the plant model either programmatically or by using the System Identification app.

This example requires a System Identification Toolbox license.

Load the measured data.

```
load dryer2
```

The variables u2 and y2, which contain the data measured for a temperature-control application, are loaded into the MATLAB® workspace. u2 is the plant input, and y2 is the plant output. The sampling period for u2 and y2 is 0.08 seconds.

Create an iddata object, dry_data, to store the measured values of the plant inputs and outputs, sampling time, channel names, etc.

```
Ts = 0.08;
dry_data = iddata(y2,u2,Ts);
```

You can optionally assign channel names and units for the input/output signals. To do so, use the InputName, OutputName, InputUnit and OutputUnit properties of an iddata object. For example:

```
dry_data.InputName = 'Power';
dry_data.OutputName = 'Temperature';
```

Detrend the measured data.

Before estimating the plant model, preprocess the measured data to increase the accuracy of the estimated model. For this example, u2 and y2 contain constant offsets that you eliminate.

```
dry_data_detrended = detrend(dry_data);
```

Estimate a linear plant model.

You can use System Identification Toolbox software to estimate a linear plant model in one of the following forms:

- State-space model
- Transfer function model
- Polynomial model
- Process model
- Grey-box model

For this example, estimate a third-order, linear state-space plant model using the detrended data.

```
plant = ssest(dry_data_detrended,3);
```

To view the estimated parameters and estimation details, at the MATLAB command prompt, type `plant`.

See Also
detrend | iddata | ssest

Related Examples
- "Identify Linear Models Using System Identification App"
- "Design Controller for Identified Plant" on page 2-54
- "Design Controller Using Identified Model with Noise Channel" on page 2-56

More About
- "Handling Offsets and Trends in Data"
- "Working with Impulse-Response Models" on page 2-58

Design Controller for Identified Plant

This example shows how to design a model predictive controller using an identified plant model. The internal plant model of the controller uses only the measured input and output of the identified model.

This example requires a System Identification Toolbox™ license.

Load the input/output data for identification.

```
load dryer2
Ts = 0.08;
```

Create an `iddata` object from the input

```
dry_data = iddata(y2,u2,Ts);
dry_data_detrended = detrend(dry_data);
```

Estimate a linear state-space plant model.

```
plant_idss = ssest(dry_data_detrended,3);
```

`plant_idss` is a third-order, identified state-space model that contains one measured input and one unmeasured noise component.

You can convert the identified model to an `ss`, `tf`, or `zpk` model. For this example, convert the identified state-space model to a numeric state-space model.

```
plant_ss = ss(plant_idss);
```

`plant_ss` contains the measured input and output of `plant_idss`. The software discards the noise component of `plant_idss` when it creates `plant_ss`.

Design a model predictive controller for the numeric plant model.

```
controller = mpc(plant_ss,Ts);
```

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming o
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1.
```

`controller` is an `mpc` object in which:

- The measured input of `plant_ss` is a manipulated variable.
- The output of `plant_ss` is a measured output.

To view the structure of the model predictive controller, type `controller` at the MATLAB® command prompt.

See Also

`mpc`

Related Examples

- "Identify Plant from Data" on page 2-52
- "Design Controller Using Identified Model with Noise Channel" on page 2-56

More About

- "About Identified Linear Models"

Design Controller Using Identified Model with Noise Channel

This example shows how to design a model predictive controller using an identified plant model with a nontrivial noise component.

This example requires a System Identification Toolbox™ license.

Load the input/output data for identification.

```
load dryer2
Ts = 0.08;
```

Create an `iddata` object from the input

```
dry_data = iddata(y2,u2,Ts);
dry_data_detrended = detrend(dry_data);
```

Estimate a linear state-space plant model.

```
plant_idss = ssest(dry_data_detrended,3);
```

`plant_idss` is a third-order, identified state-space model that contains one measured input and one unmeasured noise component.

Design a model predictive controller for the identified plant model.

```
controller = mpc(plant_idss,Ts);
```

```
-->Converting "idmodel" object to state-space.
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->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.
```

`controller` is an `mpc` object in which:

- The measured input of `plant_idss` is a manipulated variable.
- The noise component of `plant_idss` is an unmeasured disturbance.
- The output of `plant_idss` is a measured output.

To view the structure of the model predictive controller, at the MATLAB® command prompt, type `controller`.

You can change the treatment of plant input signals in one of two ways:

- Programmatic - Use the `setmpcsignals` command to set the signal types.
- MPC Designer - Specify the signal types when defining the MPC structure using an imported plant.

You can also design a model predictive controller using:

```
plant_ss = ss(plant_idss,'augmented');
controller2 = mpc(plant_ss,Ts);

-->All input signals are labeled as "Measured" or "Noise". The model was probably conve
   All "Measured" inputs will be treated as manipulated variables, all "Noise" inputs a
   Type "help setmpcsignals" for more information.
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->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.
```

When you use the `'augmented'` input argument, `ss` creates two input groups, `Measured` and `Noise`, for the measured and noise inputs of `plant_idss`. `mpc` handles the measured and noise components of `plant_ss` and `plant_idss` identically.

See Also
`mpc`

Related Examples
- "Identify Plant from Data" on page 2-52
- "Design Controller for Identified Plant" on page 2-54

More About
- "About Identified Linear Models"

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* (see [1] for example), are easy to determine from plant data ([2] and [3]) and have intuitive appeal.

You use the `impulseest` function to estimate an FIR model from measured data. The function returns an identified transfer function model, `idtf`. To design a model predictive controller for the 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. This result is particularly an issue for MIMO systems. For example, the estimator design can be affected by numerical precision issues with high-order plants.

Model predictive controllers work best with low-order parametric models. See [4] for an example. Therefore, to design a model predictive controller using measured plant data, you can use a parametric estimator, such as `ssest`. Then estimate a low-order parametric plant model. Alternatively, you can initially identify a nonparametric model using the data and then estimate a low-order parametric model for the nonparametric model's response. See [5] for an example.

References

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

[2] 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.

[3] 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.

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

[5] 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
impulseest | ssest

Related Examples

- "Identify Plant from Data" on page 2-52
- "Design Controller for Identified Plant" on page 2-54
- "Design Controller Using Identified Model with Noise Channel" on page 2-56

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.

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

3

Designing Controllers Using MPC Designer

Design Controller Using MPC Designer

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

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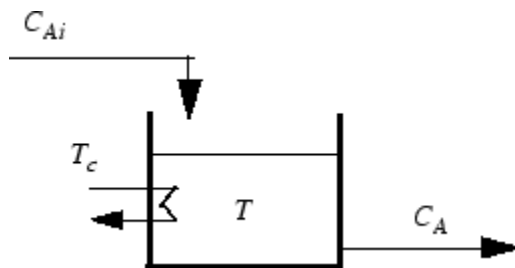
Measurement of reagent concentrations is often difficult. For this example, assume that:

- 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.
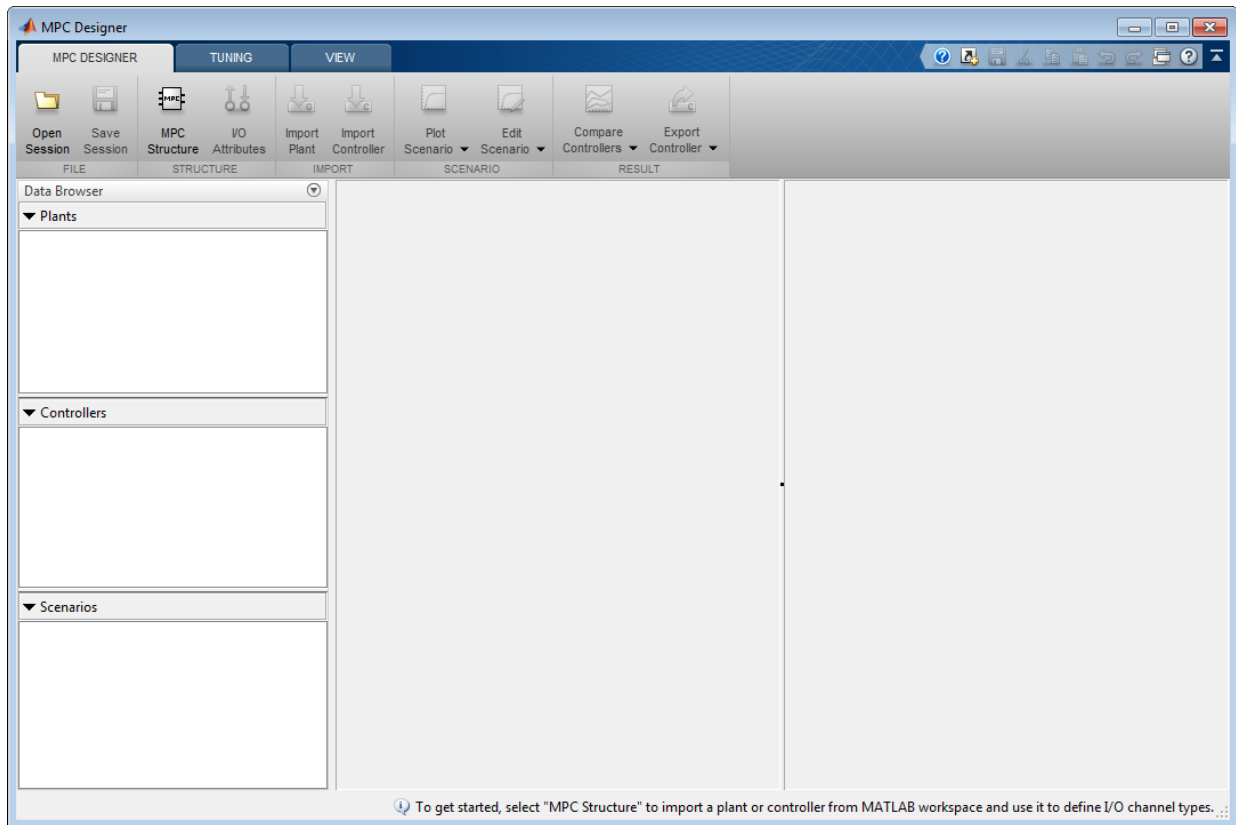
Define Plant Model

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);
```

Open MPC Designer App

mpcDesigner

Import Plant and Define MPC Structure

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.

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. Since the CSTR plant is a continuous-time model, the default sample time of the controller is 1 second.
- scenario1 — Default simulation scenario.

The app runs the default simulation scenario and updates the **Input Response** and **Output Response** plots.
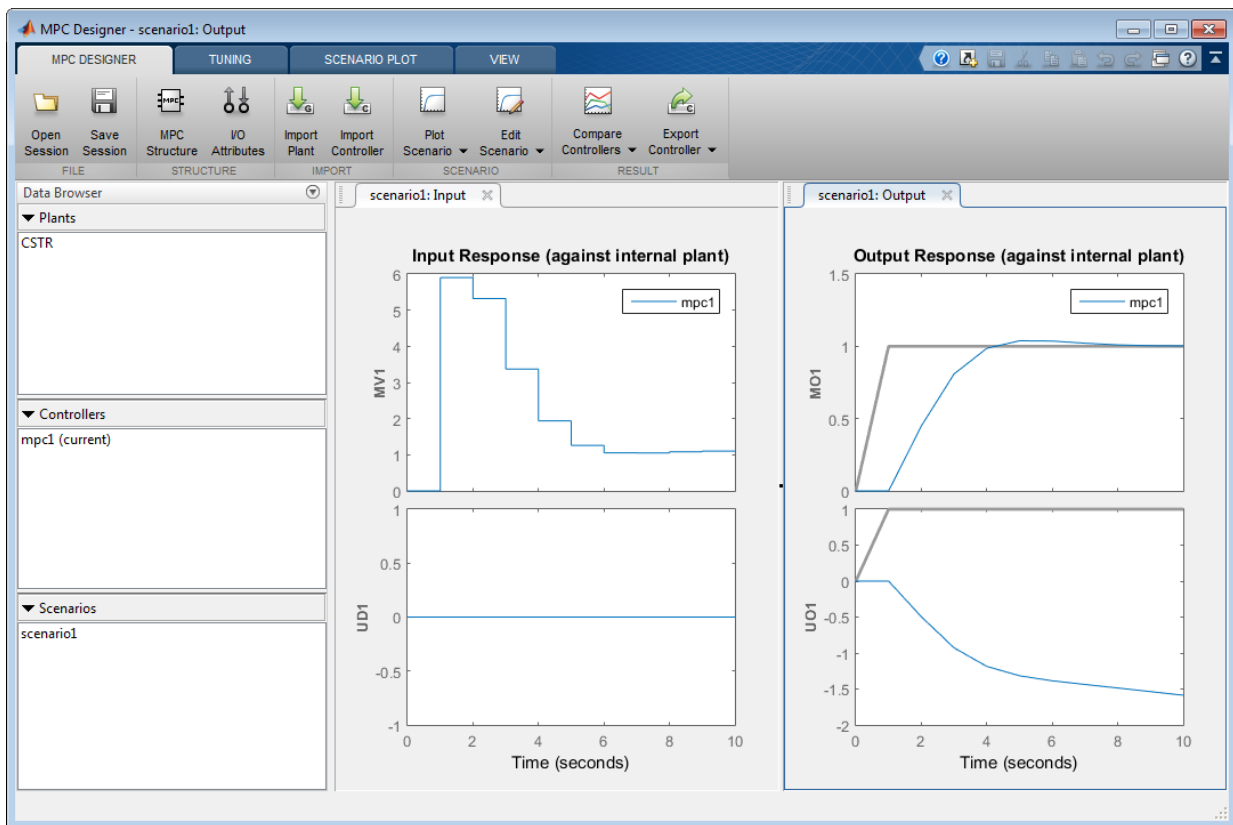
Tip To view the response plots side-by-side, on the **View** tab, in the **Tiles** section, click **Left/Right**.
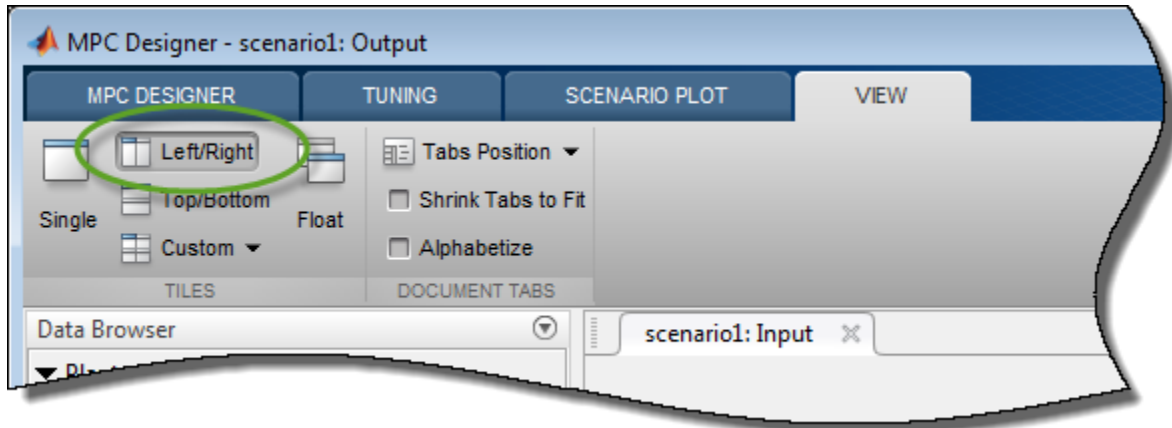
Once you define the MPC structure, you cannot change it within the current MPC Designer session. To use a different channel configuration, start a new session of the app.

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.

Input and Output Channel Specifications   ✕

Plant Inputs

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

Plant Outputs

Channel	Type	Name	Unit	Nominal Value	Scale Factor
y(1)	MO	T	deg C	0	1
y(2)	UO	CA	kgmol/m^3	0	1

OK | Apply | Cancel | Help

Click **OK**.

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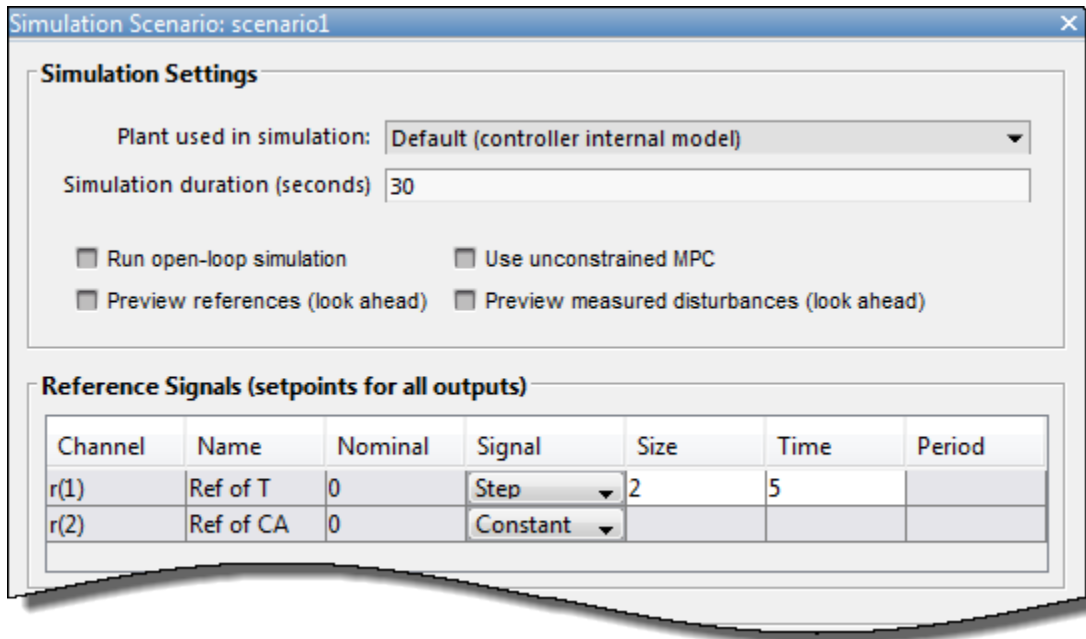
Configure Step Response Simulation Scenario

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.

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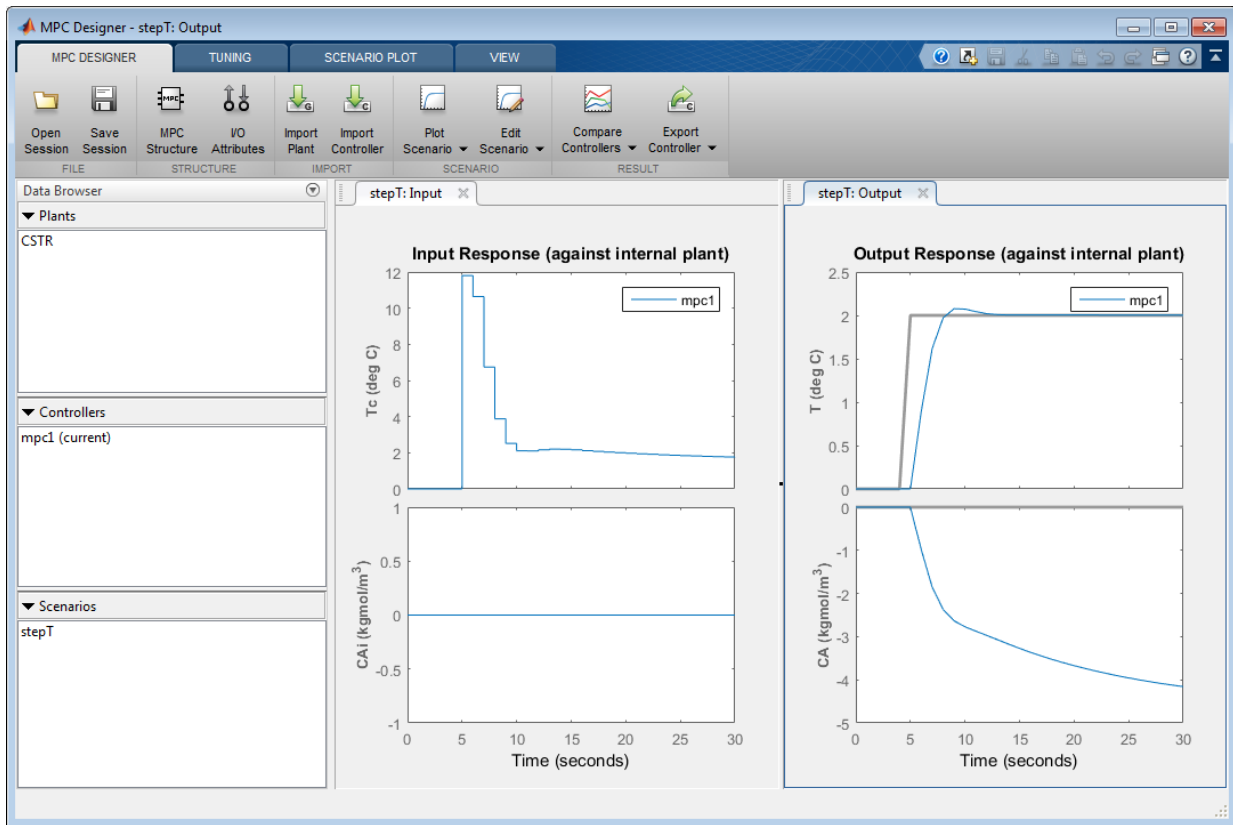
The default scenario is configured to simulate a step change of 2 degrees in the reactor temperature, *T* at a time of 5 seconds.

Click **OK**.

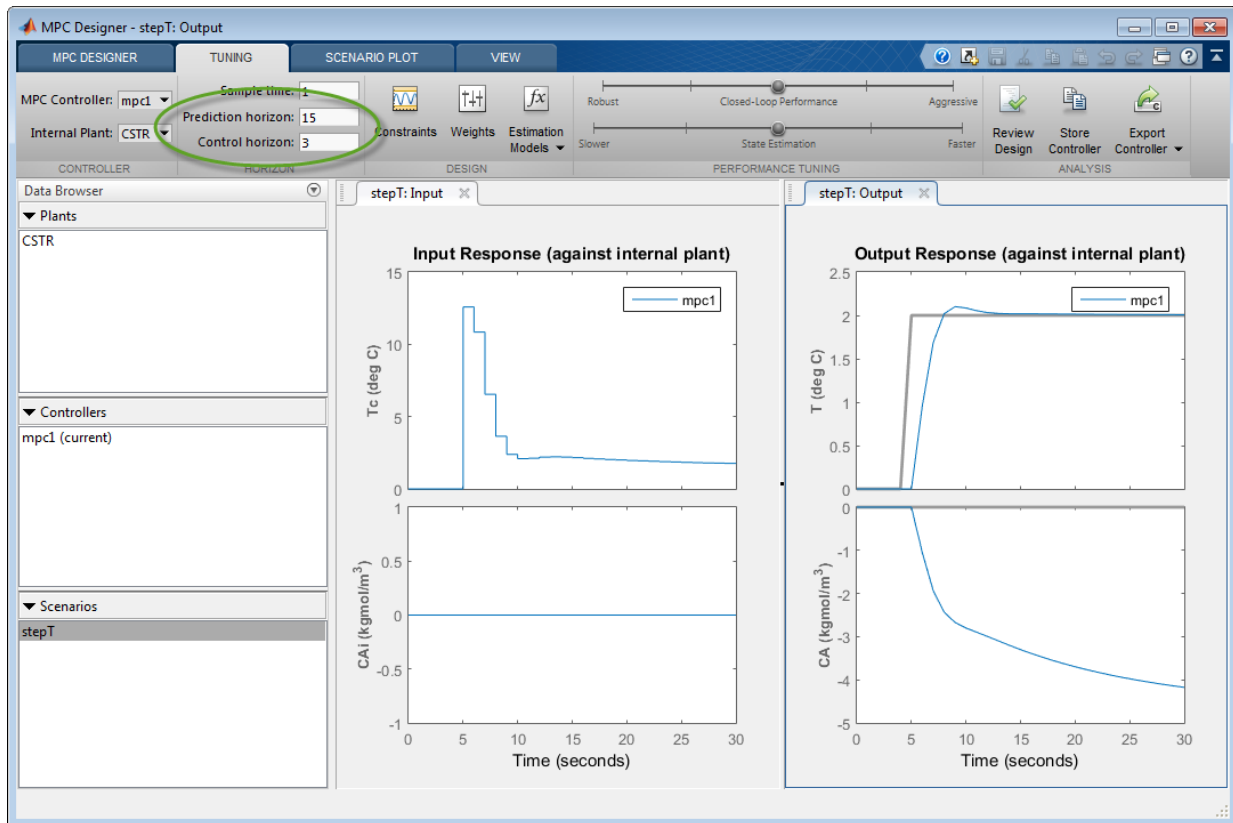The response plots update to reflect the new simulation scenario configuration.

In the **Data Browser**, in the **Scenarios** section, double-click scenario1, and rename the scenario as stepT.

Configure Controller Horizons

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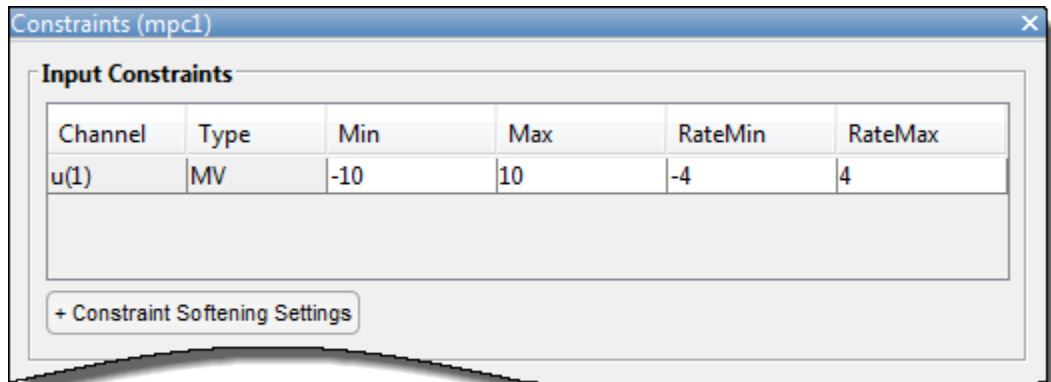
The response plots update to reflect the new horizons. The **Input Response** plot shows that the control actions for the manipulated variable violate the required coolant temperature constraints.

Define Manipulated Variable Input Constraints

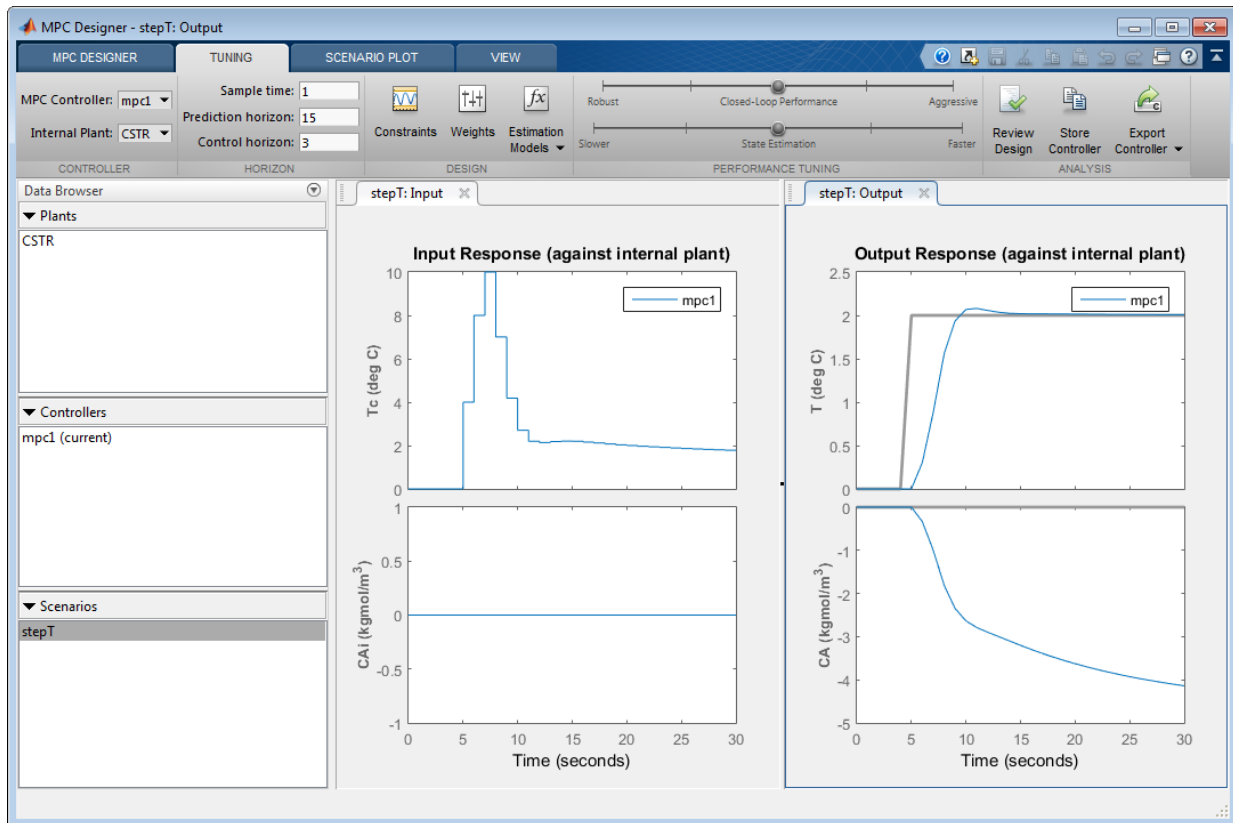In the **Design** section, click **Constraints**.

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.

Constraints (mpc1)					✕
Input Constraints					
Channel	Type	Min	Max	RateMin	RateMax
u(1)	MV	-10	10	-4	4

+ Constraint Softening Settings

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.
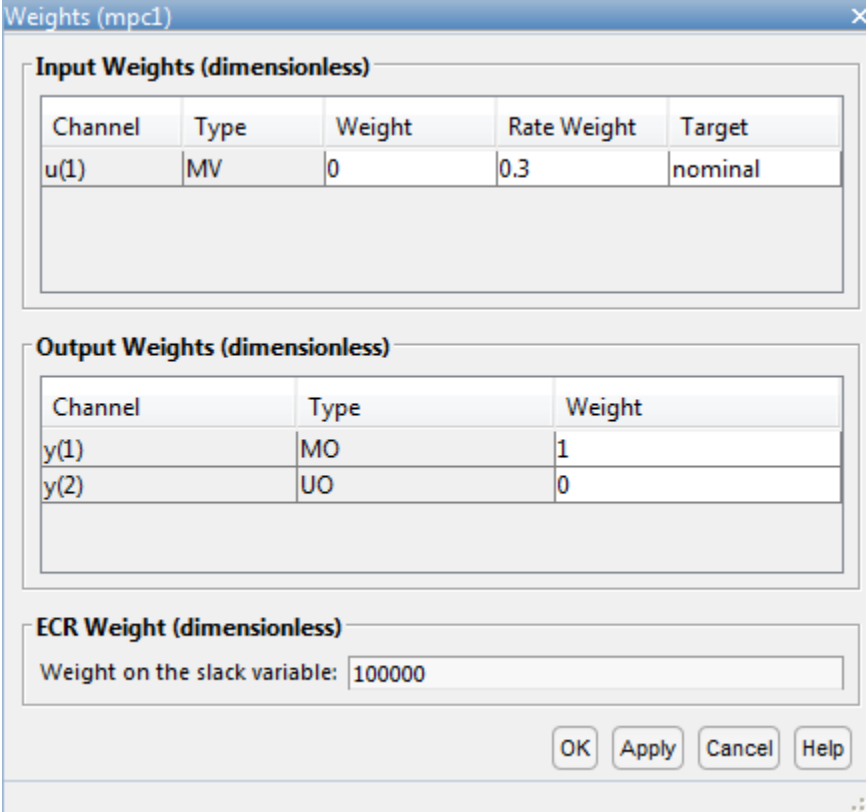
Specify Controller Tuning Weights

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.
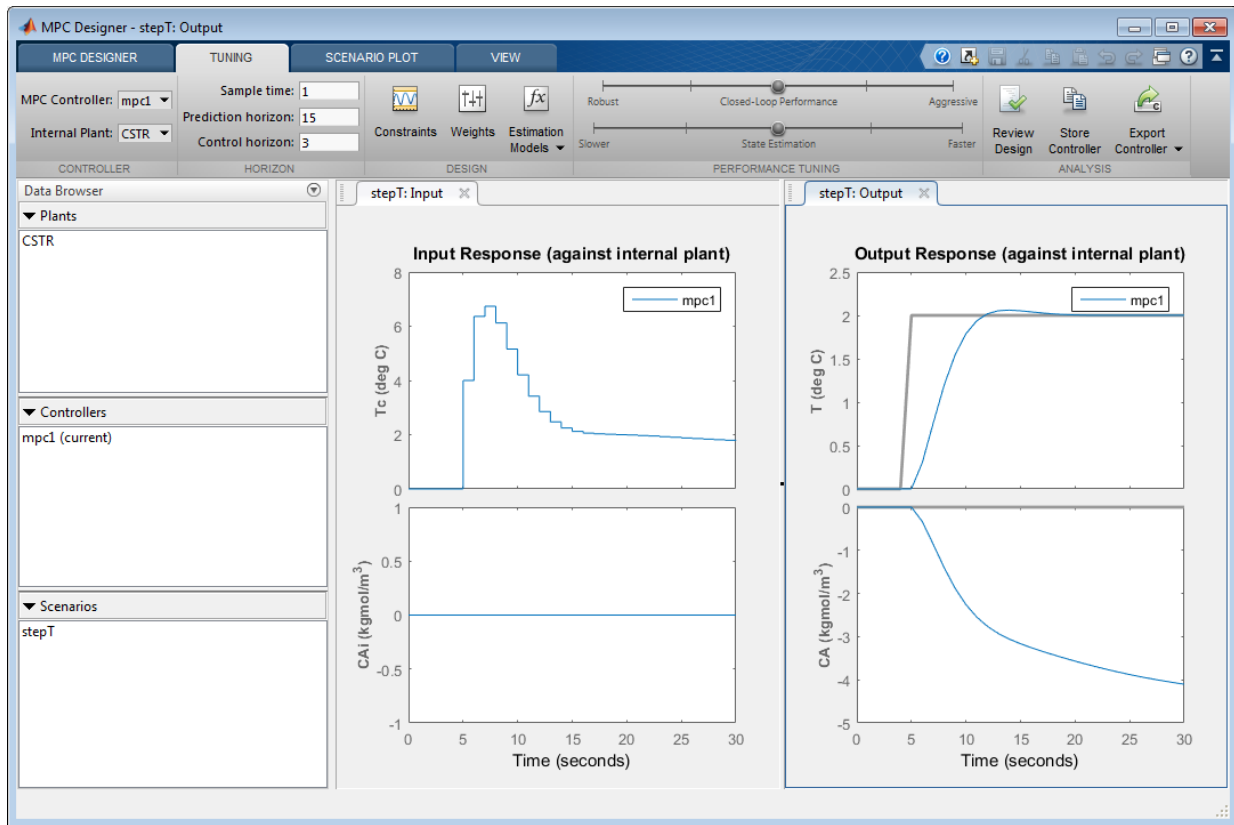
Weights (mpc1)

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 reduces the manipulated variable rate weight and increases the output variable weight, producing a more robust controller.
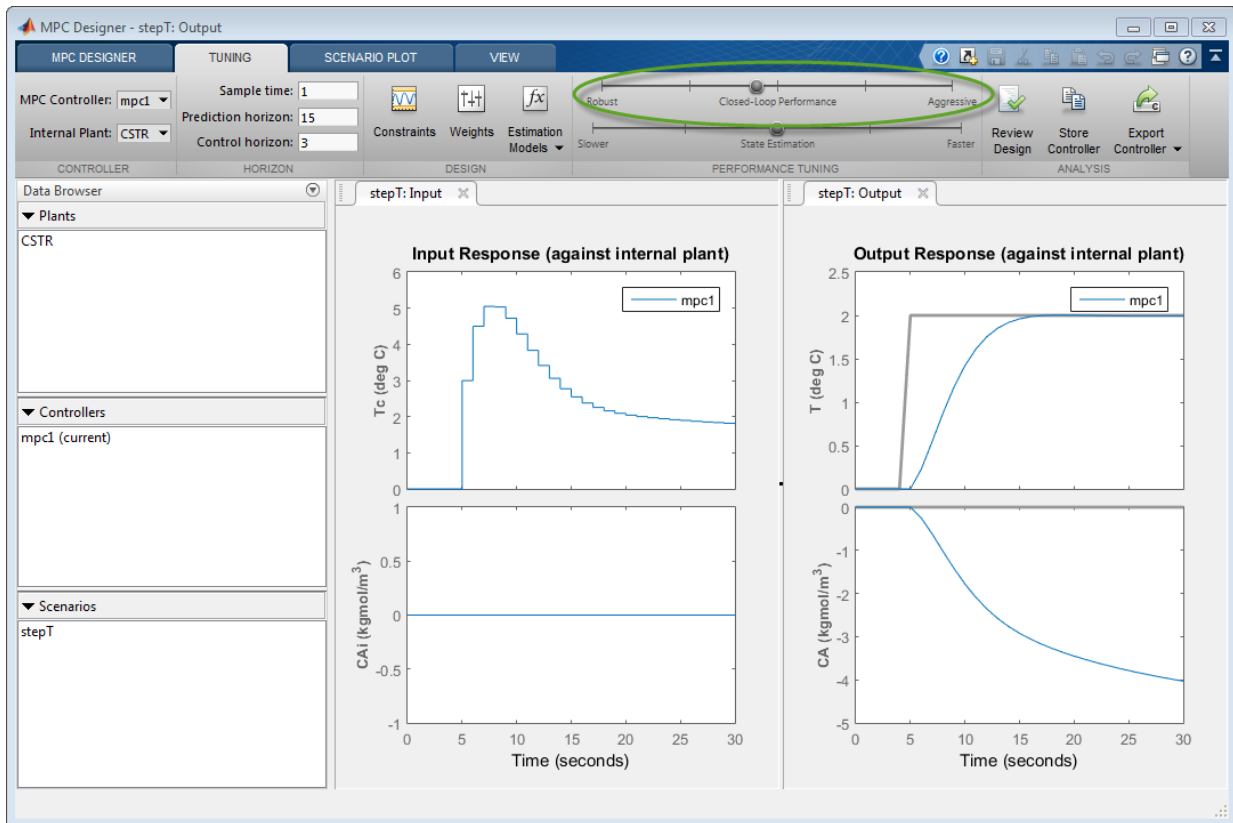
Test Controller Disturbance Rejection

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.

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.

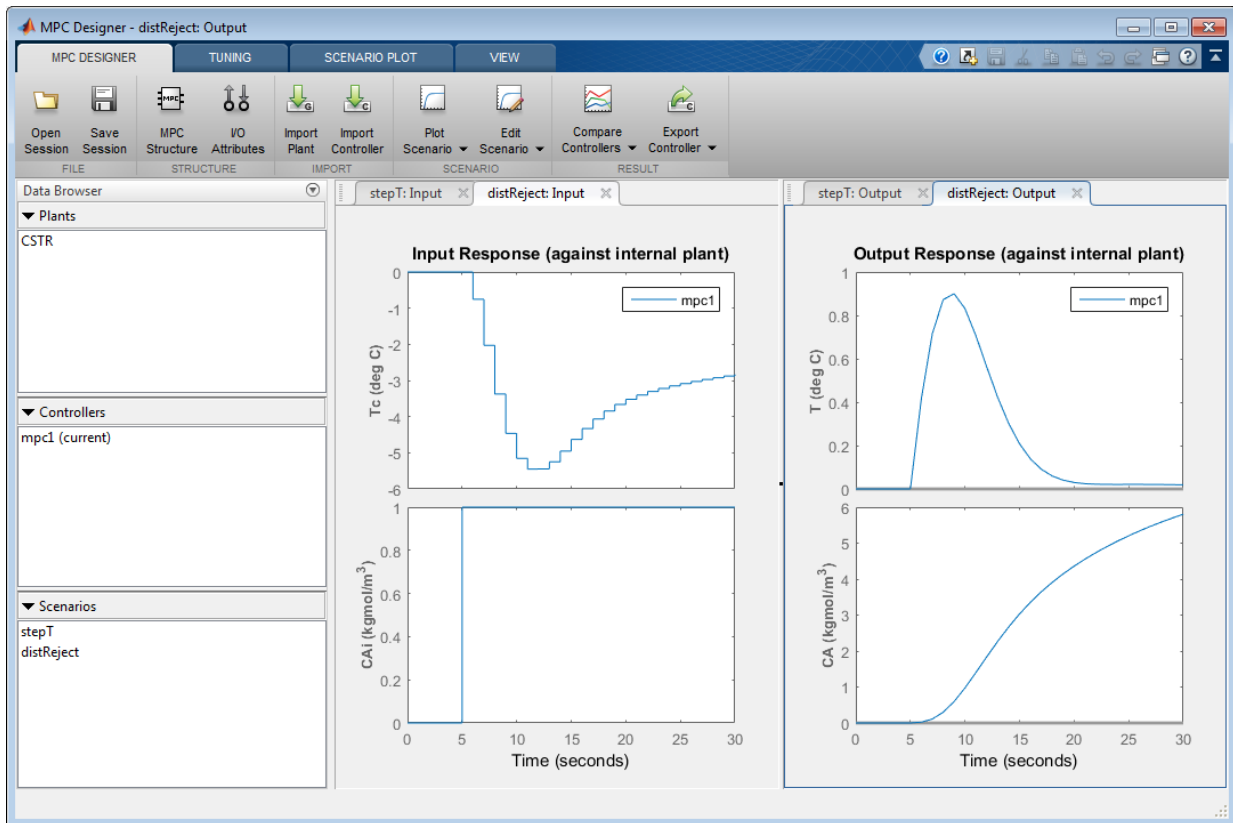In the **Unmeasured Disturbances** table, in the **Signal** drop-down list, select Step.

In the **Time** column, specify a step time of 5 seconds.
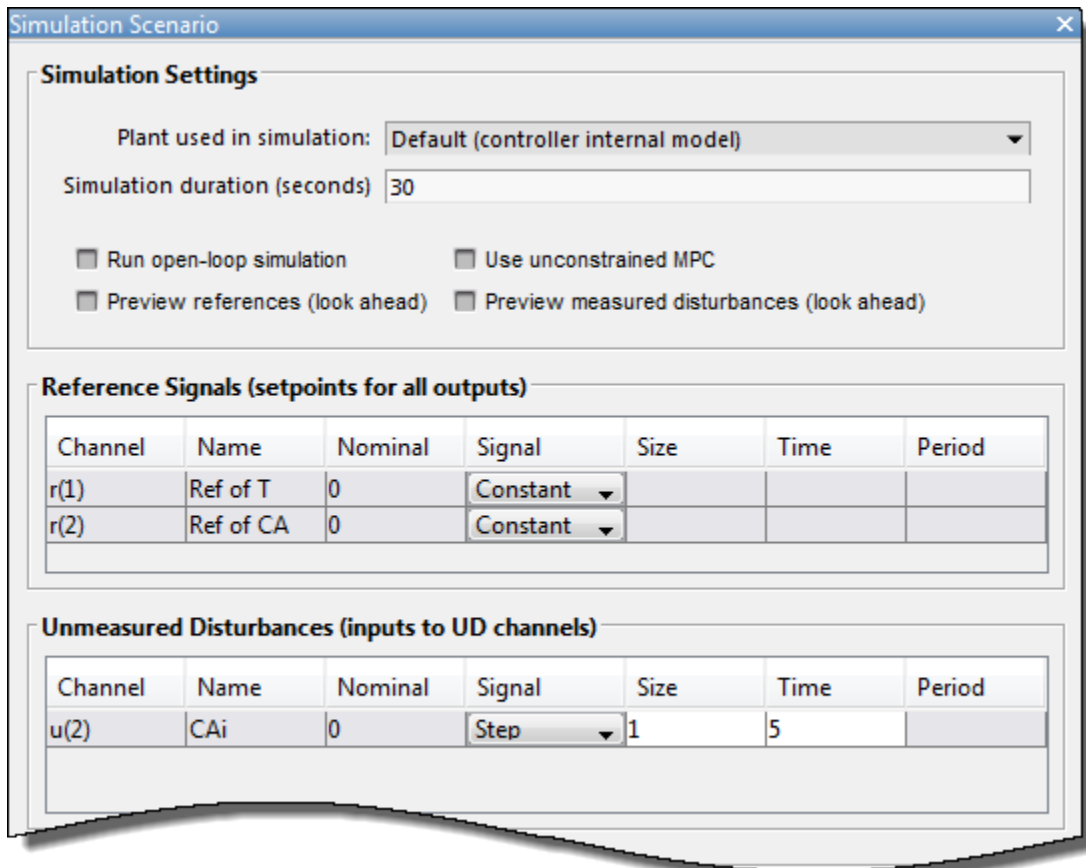
Click **OK**.

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, double-click `NewSceanrio`, and rename it `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/m^3.

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/m^3. 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.
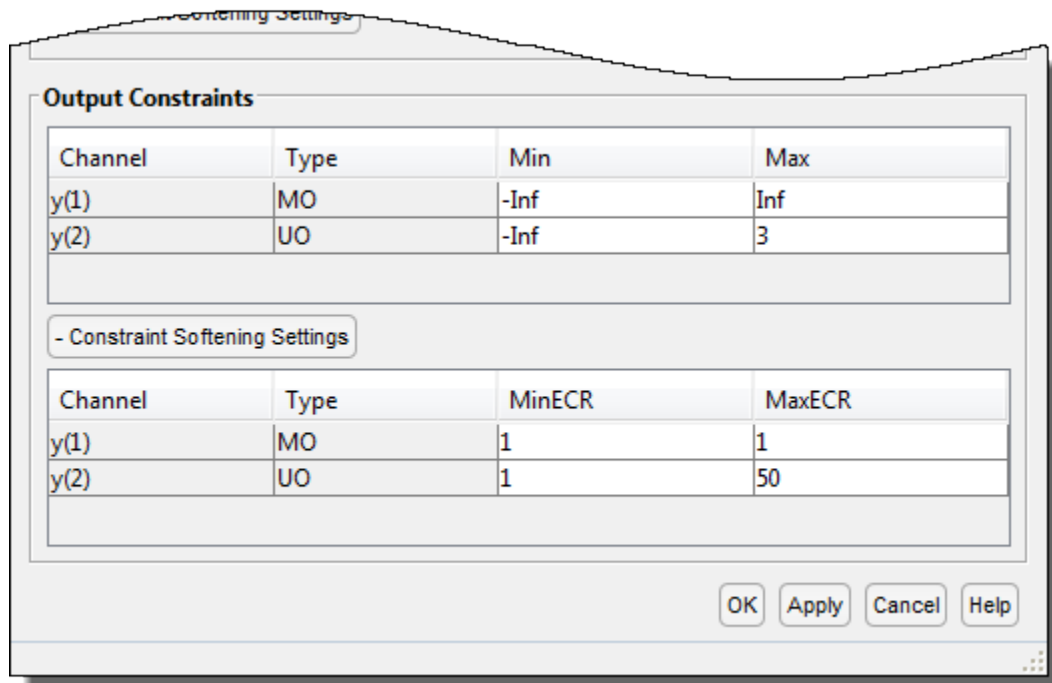
Click **OK**.
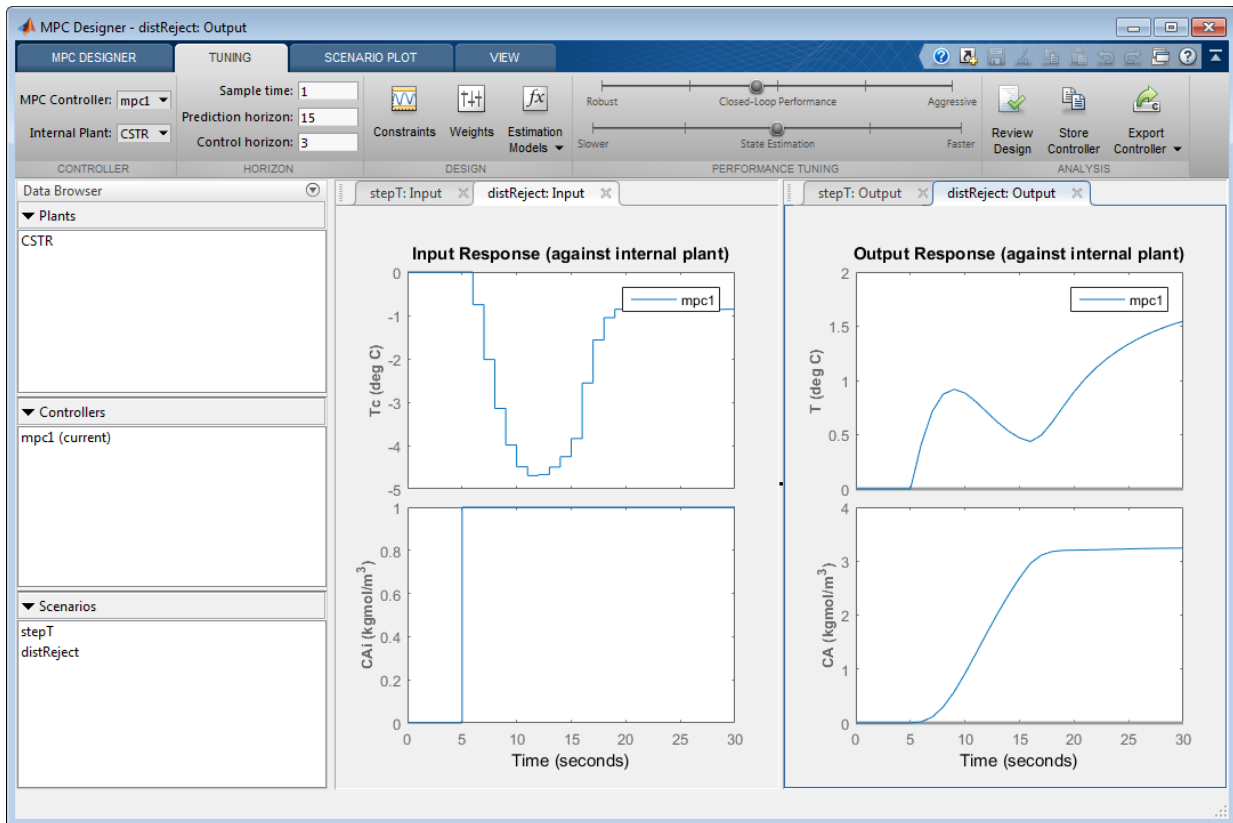
In the **Output Response** plots, once the reactor concentration, **CA**, approaches 3 kgmol/m³, the reactor temperature, **T**, starts to increase. Since there is only one manipulated variable, the controller makes a compromise between the two competing control objectives: Temperature control and constraint 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.

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

Related Examples
- "Design MPC Controller in Simulink" on page 5-2

More About
- "Specifying Constraints"
- "Tuning Weights"

Test Controller Robustness

This example shows how to test the sensitivity of your model predictive controller to prediction errors using simulations.

It is good practice to test the robustness of your controller to prediction errors. Classical phase and gain margins are one way to quantify robustness for a SISO application. Robust Control Toolbox™ software provides sophisticated approaches for MIMO systems. It can also be helpful to run simulations.

Define Plant Model

For this example, use the CSTR model described in "Design Controller Using MPC Designer" on page 3-2.

```
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);
```

Specify the signal names and signal types for the plant.

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

Open MPC Designer App

Open the MPC Designer app, and import the plant model.

```
mpcDesigner(CSTR)
```

The app imports the plant model and adds it to the **Data Browser**. It also creates a default controller and a default simulation scenario.

Design Controller

Typically, you would design your controller by specifying scaling factors, defining constraints, and adjusting tuning weights. For this example, modify the controller sample time, and keep the other controller settings at their default values.

In the MPC Designer app, on the **Tuning** tab, in the **Horizon** section, specify a **Sample time** of 0.25 seconds.
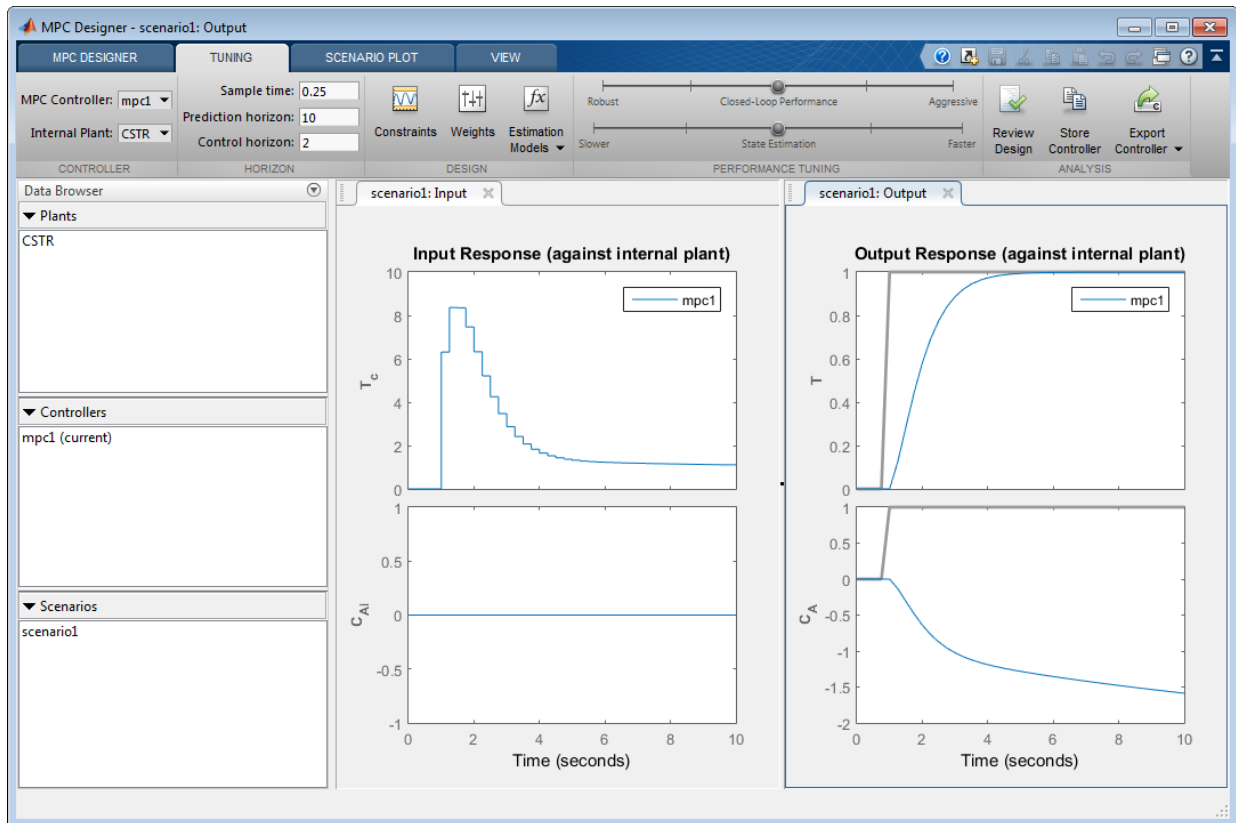
The **Input Response** and **Output Response** plots update to reflect the new sample time.

Configure Simulation Scenario

To test controller setpoint tracking and unmeasured disturbance rejection, modify the default simulation scenario.

In the **Data Browser**, in the **Scenarios** sections, right-click scenario1, and select **Edit**.

In the Simulation Scenario dialog box, specify a **Simulation duration** of 50 seconds.

In the **Reference Signals** table, keep the default `Ref of T` setpoint configuration, which simulates a unit-step change in the reactor temperature.

To hold the concentration setpoint at its nominal value, in the second row, in the **Signal** drop-down list, select `Constant`.

Simulate a unit-step unmeasured disturbance at a time of 25 seconds. In the **Unmeasured Disturbances** table, in the **Signal** drop-down list, select `Step`, and specify a **Time** of `25`.

Simulation Scenario: scenario1

Simulation Settings

Plant used in simulation: Default (controller internal model)

Simulation duration (seconds) 50

☐ Run open-loop simulation   ☐ Use unconstrained MPC
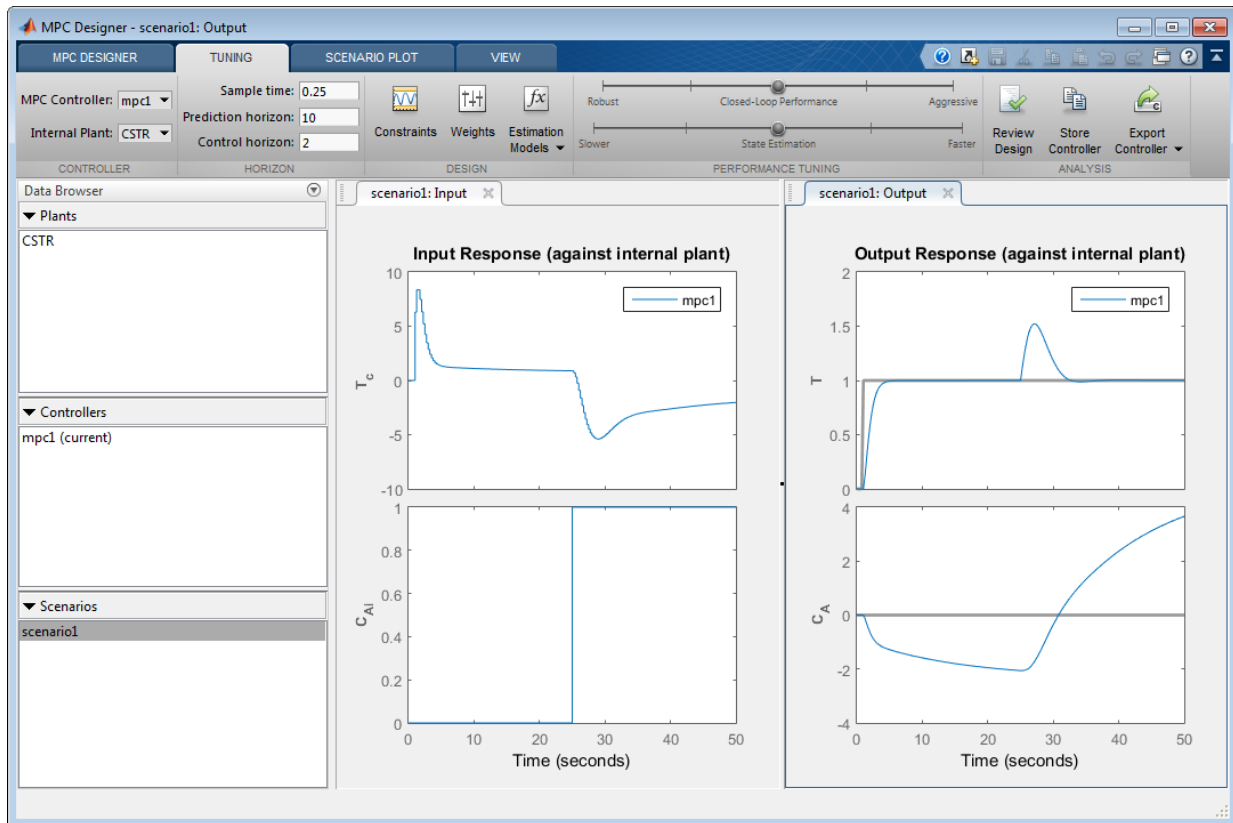☐ Preview references (look ahead)   ☐ Preview measured disturbances (look ahead)

Reference Signals (setpoints for all outputs)

Channel	Name	Nominal	Signal	Size	Time	Period
r(1)	Ref of T	0	Step	1	1	
r(2)	Ref of C_A	0	Constant			

Unmeasured Disturbances (inputs to UD channels)

Channel	Name	Nominal	Signal	Size	Time	Period
u(2)	C_A_i	0	Step	1	25	

Click **OK**.

The app runs the simulation scenario, and updates the response plots to reflect the new simulation settings. For this scenario, the internal model of the controller is used in the simulation. Therefore, the simulation results represent the controller performance when there are no prediction errors.

Define Perturbed Plant Models

Suppose that you want to test the sensitivity of your controller to plant changes that modify the effect of the coolant temperature on the reactor temperature. You can simulate such changes by perturbing element `B(2,1)` of the CSTR input-to-state matrix.

In the MATLAB Command Window, specify the perturbation matrix.

```
dB = [0 0;0.05 0];
```
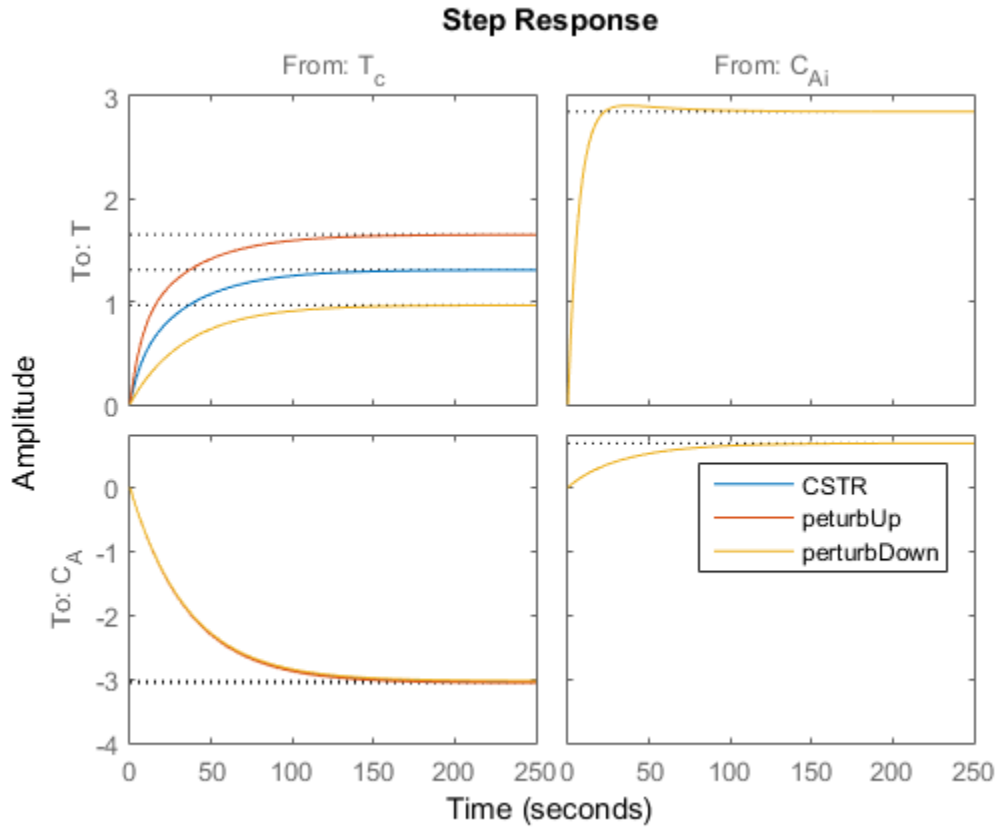
Create the two perturbed plant models.

```
perturbUp = CSTR;
perturbUp.B = perturbUp.B + dB;

perturbDown = CSTR;
perturbDown.B = perturbDown.B - dB;
```

Examine Step Responses of Perturbed Plants

To examine the effects of the plant perturbations, plot the plant step responses.

```
step(CSTR,perturbUp,perturbDown)
legend('CSTR','peturbUp','perturbDown')
```
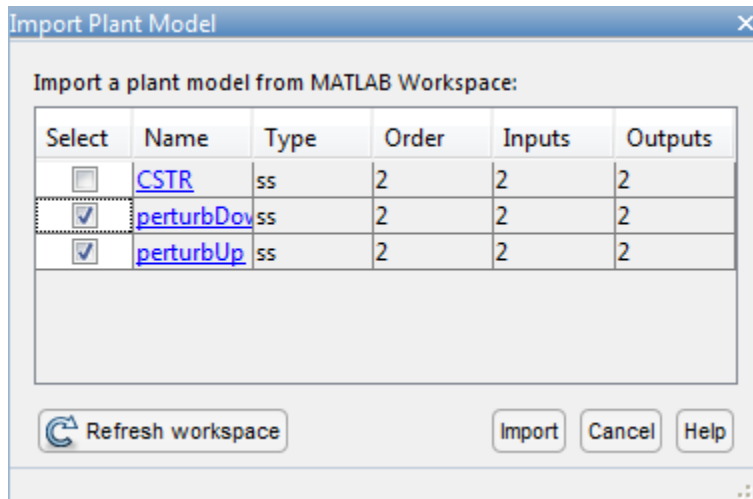
Perturbing element `B(2,1)` of the CSTR plant changes the magnitude of the response of the reactor temperature, T, to changes in the coolant temperature, `Tc`.

Import Perturbed Plants

In the MPC Designer app, on the **MPC Designer** tab, in the **Import** section, click **Import Plant**.

In the Import Plant Model dialog box, select the `perturbUp` and `perturbDown` models.

Click **Import**.

The app imports the models and adds them to the **Data Browser**.

Define Perturbed Plant Simulation Scenarios

Create two simulation scenarios that use the perturbed plant models.

In the **Data Browser**, in the **Scenarios** section, double-click scenario1, and rename it accurate.

Right-click accurate, and click **Copy**. Rename accurate_Copy to errorUp.

Right-click errorUp, and select **Edit**.

In the Simulation Scenario dialog box, in the **Plant used in simulation** drop-down list, select perturbUp.
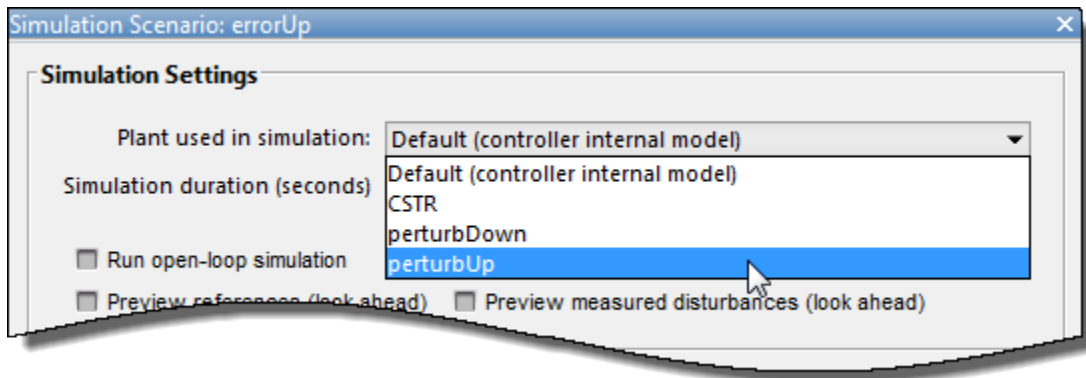
Click **OK**.

Repeat this process for the second perturbed plant.

Copy the `accurate` scenario and rename it to `errorDown`.

Edit `errorDown`, selecting the `perturbDown` plant.
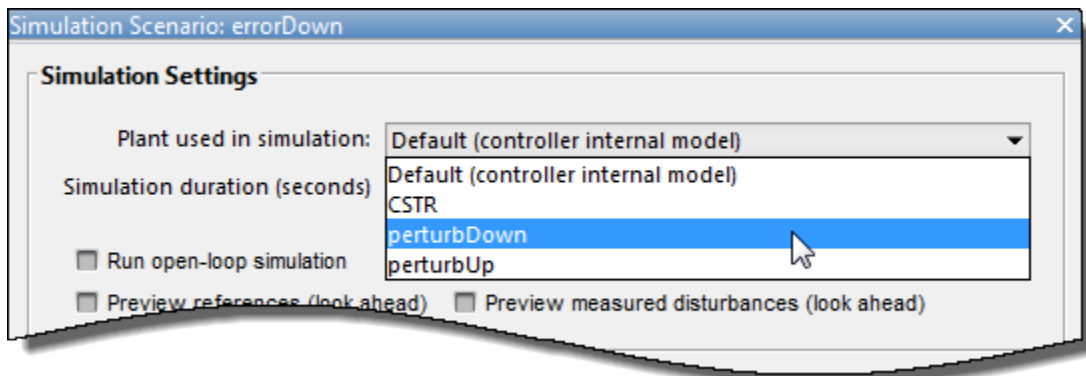
Examine `errorUp` Simulation Response

On the **MPC Designer** tab, in the **Scenario** section, click **Plot Scenario** > **errorUp**.
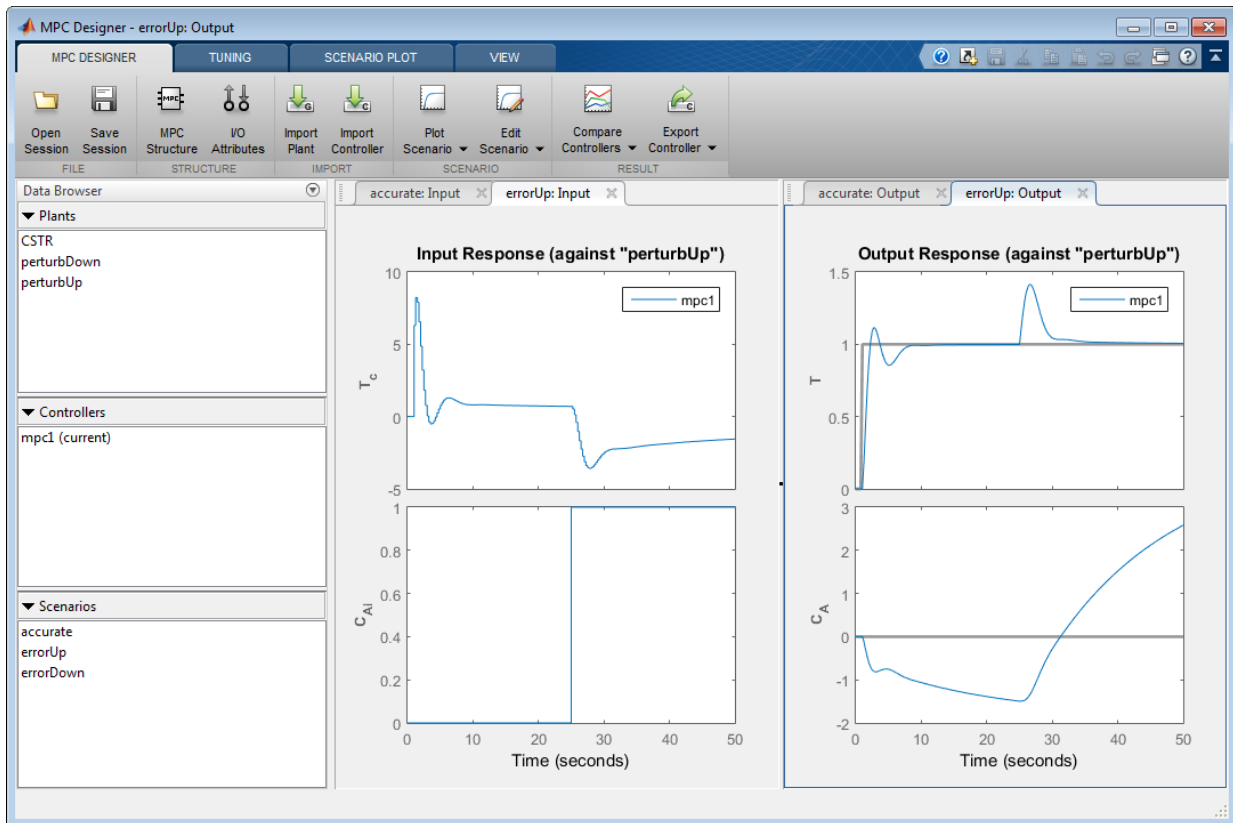
The app creates the **errorUp: Input** and **errorUp: Output** tabs, and displays the simulation response.

To view the `accurate` and `errorUp` responses side-by-side, drag the **accurate: Output** tab into the left plot panel.

The perturbation creates a plant, `perturbUp`, that responds faster to manipulated variable changes than the controller predicts. On the **errorUp: Output** tab, in the **Output Response** plot, the **T** setpoint step response has about 10% overshoot with a longer settling time. Although t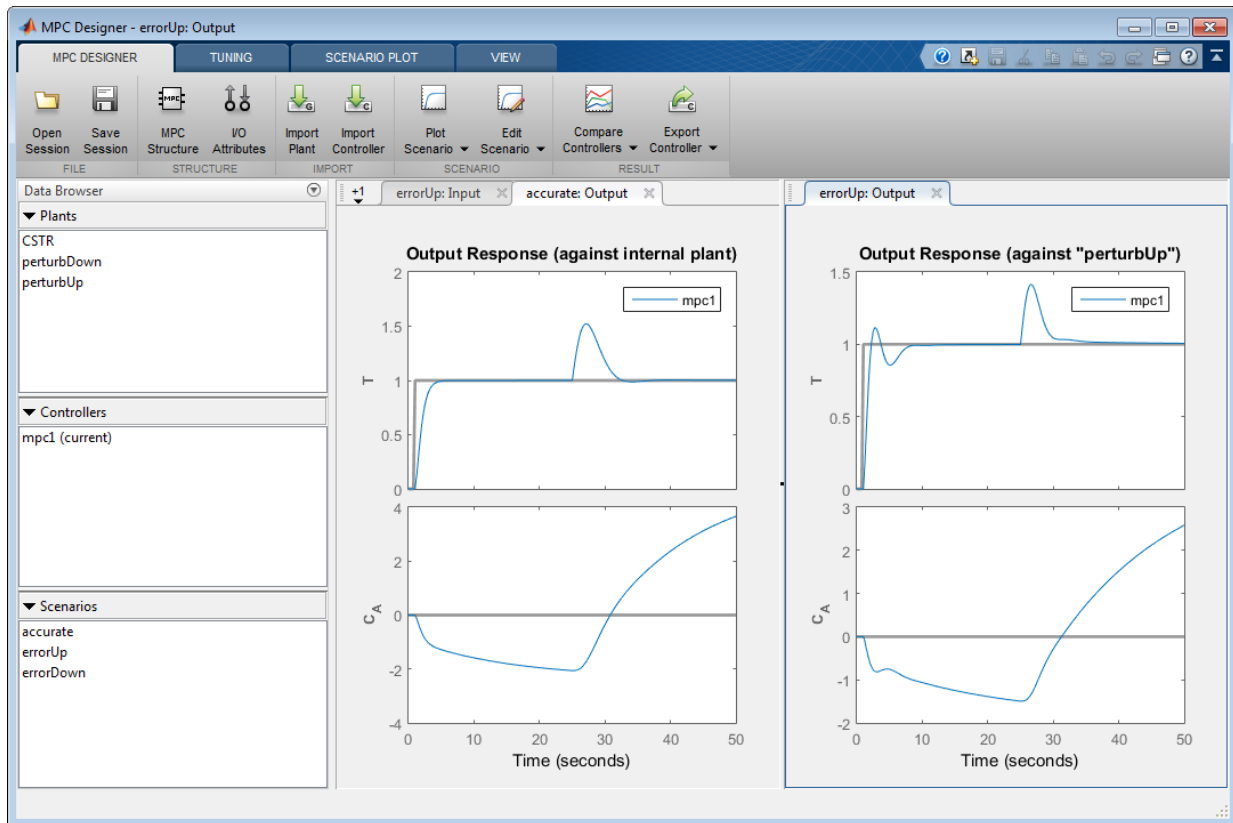his response is worse than the response of the `accurate` simulation, it is still acceptable. The faster plant response leads to a smaller peak error due to the unmeasured disturbance. Overall, the controller is able to control the `perturbUp` plant successfully despite the internal model prediction error.

Examine `errorDown` Simulation Response

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

The app creates the **errorDown: Input** and **errorDown: Output** tabs, and displays the simulation response.

To view the `accurate` and `errorDown` responses side-by-side, click the **accurate: Output** tab in the left display panel.
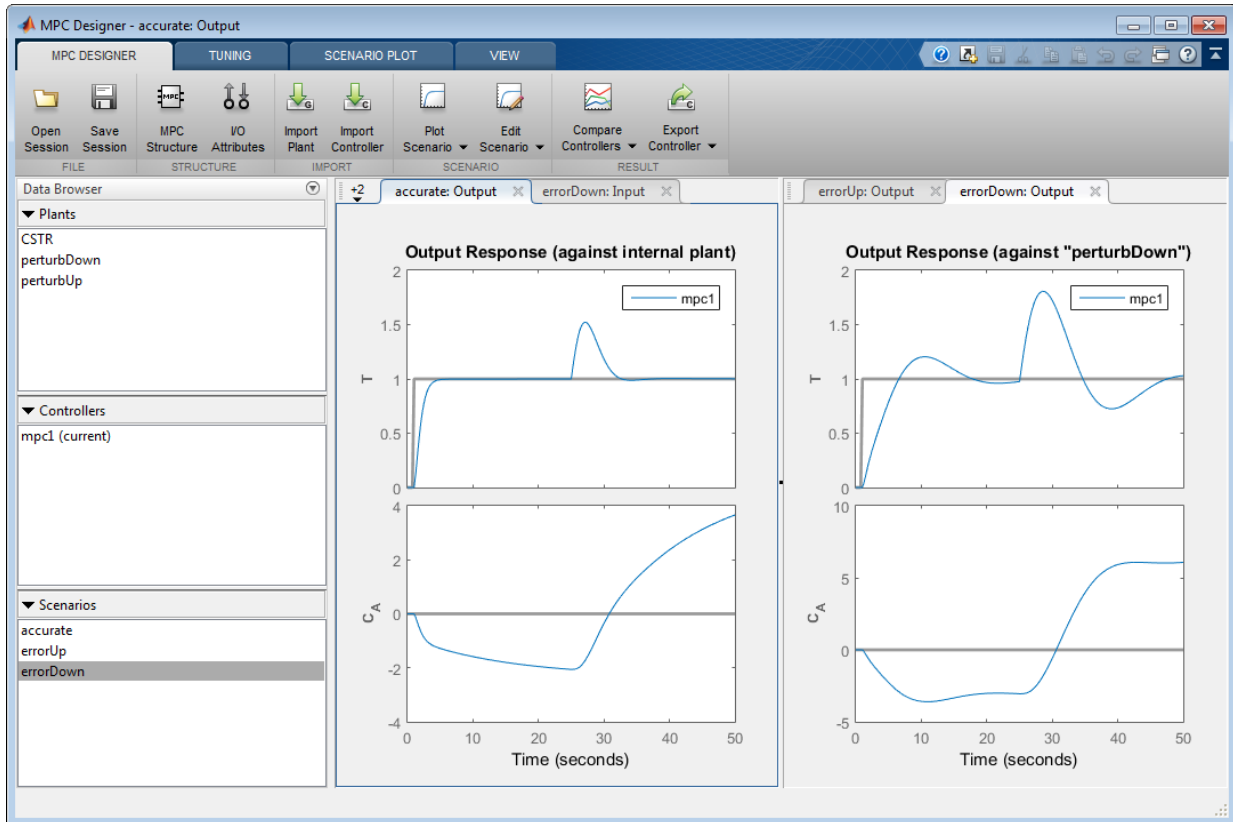

The perturbation creates a plant, `perturbDown`, that responds slower to manipulated variable changes than the controller predicts. On the **errorDown: Output** tab, in the **Output Response** plot, the setpoint tracking and disturbance rejection are worse than for the unperturbed plant.

Depending on the application requirements and the real-world potential for such plant changes, the degraded response for the `perturbDown` plant may require modifications to the controller design.

See Also
`mpc` | MPC Designer

Related Examples
- "Design Controller Using MPC Designer" on page 3-2
- "Test an Existing Controller" on page 5-23

Design MPC Controller for Plant with Delays

This example shows how to design an MPC controller for a plant with delays using the MPC Designer app.

Plant Model

An example of a plant with delays is the distillation column model:

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

Outputs y_1 and y_2 represent measured product purities. The model consists of six transfer functions, one for each input/output pair. Each transfer function is a first-order system with a delay. The longest delay in the model is 8.1 minutes.

Define Plant Model

Specify the individual transfer functions for each input/output pair. For example, g12 is the transfer function from input u_2 to output y_1.

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

Specify Input and Output Signal Names

```
DC.InputName = {'Reflux Rate','Steam Rate','Feed Rate'};
DC.OutputName = {'Distillate Purity','Bottoms Purity'};
```

Alternatively, you can specify the signal names in the MPC Designer app, on the **MPC Designer** tab, by clicking **I/O Attributes**.

Define Signal Configuration

Specify the third input, the feed rate, as a measured disturbance (MD).

```
DC = setmpcsignals(DC,'MD',3);
```

Since they are not explicitly specified in `setmpcsignals`, all other input signals are configured as manipulated variables (MV), and all output signals are configured as measured outputs (MO) by default.

Open MPC Designer App

Open the MPC Designer app importing the plant model.

```
mpcDesigner(DC)
```
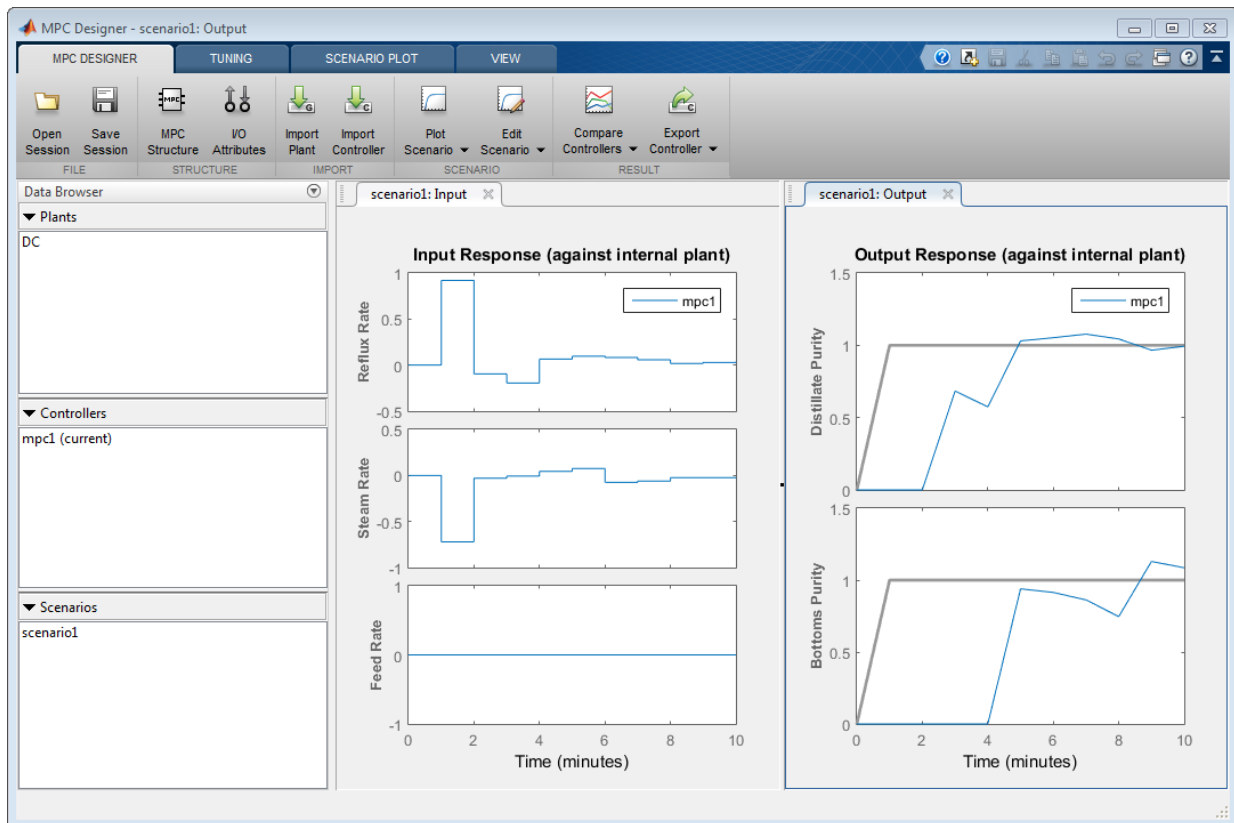

When launched with a continuous-time plant model, such as `DC`, the default controller sample time is 1 in the time units of the plant. If the plant is discrete time, the controller sample time is the same as the plant sample time.

The MPC Designer app imports the specified plant to the **Data Browser**. The following are also added to the **Data Browser**:

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

The app runs the simulation scenario and generates input and output response plots.

Specify Prediction and Control Horizons

For a plant with delays, it is good practice to specify the prediction and control horizons such that

$$P - M \gg t_{d,max} \, / \, \Delta t$$

where,

- P is the prediction horizon.
- M is the control horizon.
- $t_{d,max}$ is the maximum delay, which is `8.1` minutes for the `DC` model.
- Δt is the controller **Sample time**, which is `1` minute by default.

On the **Tuning** tab, in the **Horizon** section, specify a **Prediction horizon** of `30` and a **Control horizon** of `5`.
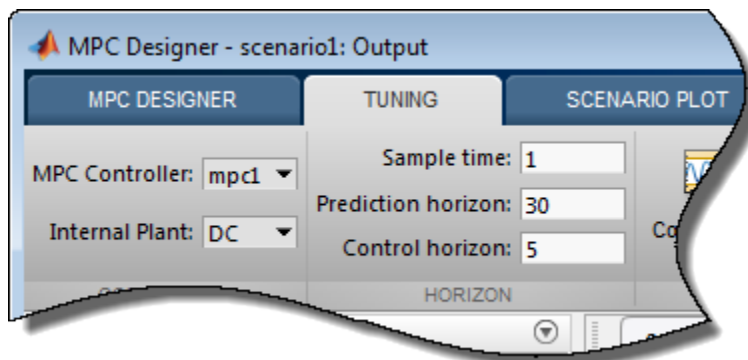
After you change the horizons, the **Input Response** and **Output Response** plots for the default simulation scenario are automatically updated.

Simulate Controller Step Responses

On the **MPC Designer** tab, in the **Scenario** section, click **Edit Scenario > scenario1**. Alternatively, in the **Data Browser**, right-click scenario1 and select **Edit**.

In the Simulation Scenario dialog box, specify a **Simulation duration** of 50 minutes.

In the **Reference Signals** table, in the **Signal** drop-down list, select Step for both outputs to simulate step changes in their setpoints.

Specify a step **Time** of 0 for reference **r(1)**, the distillate purity, and a step time of 25 for **r(2)**, the bottoms purity.
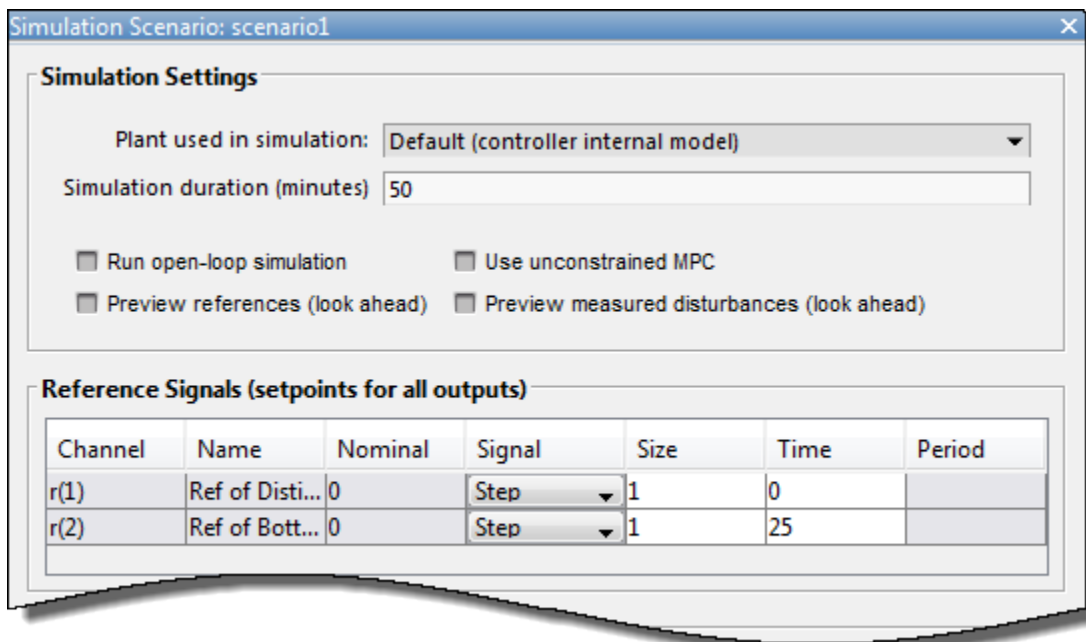
Click **OK**.

The app runs the simulation with the new scenario settings and updates the input and output response plots.

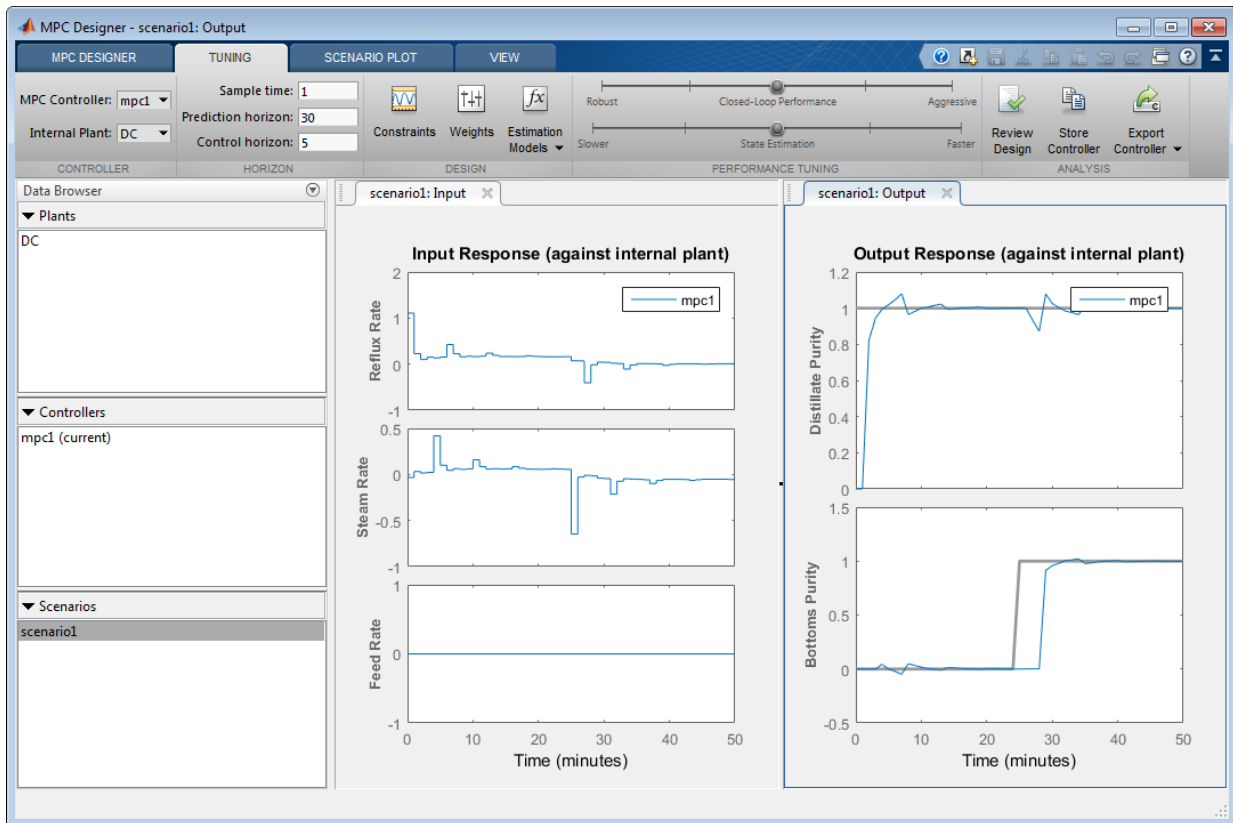The **Input Response** plots show the optimal control moves generated by the controller. The controller reacts immediately in response to the setpoint changes, changing both manipulated variables. However, due to the plant delays, the effects of these changes are not immediately reflected in the **Output Response** plots. The **Distillate Purity** output responds after 1 minute, which corresponds to the minimum delay from $g11$ and $g12$. Similarly, the **Bottoms Purity** output responds 3 minutes after the step change, which corresponds to the minimum delay from $g21$ and $g22$. After the initial delays, both signals reach their setpoints and settle quickly. Changing either output setpoint disturbs the response of the other output. However, the magnitudes of these interactions are less than 10% of the step size.

Additionally, there are periodic pulses in the manipulated variable control actions as the controller attempts to counteract the delayed effects of each input on the two outputs.

Improve Performance Using Manipulated Variable Blocking

Use manipulated variable blocking to divide the prediction horizon into blocks, during which manipulated variable moves are constant. This technique produces smoother manipulated variable adjustments with less oscillation and smaller move sizes.

To use manipulated variable blocking, on the **Tuning** tab, specify the **Control horizon** as a vector of block sizes, [5 5 5 5 10].
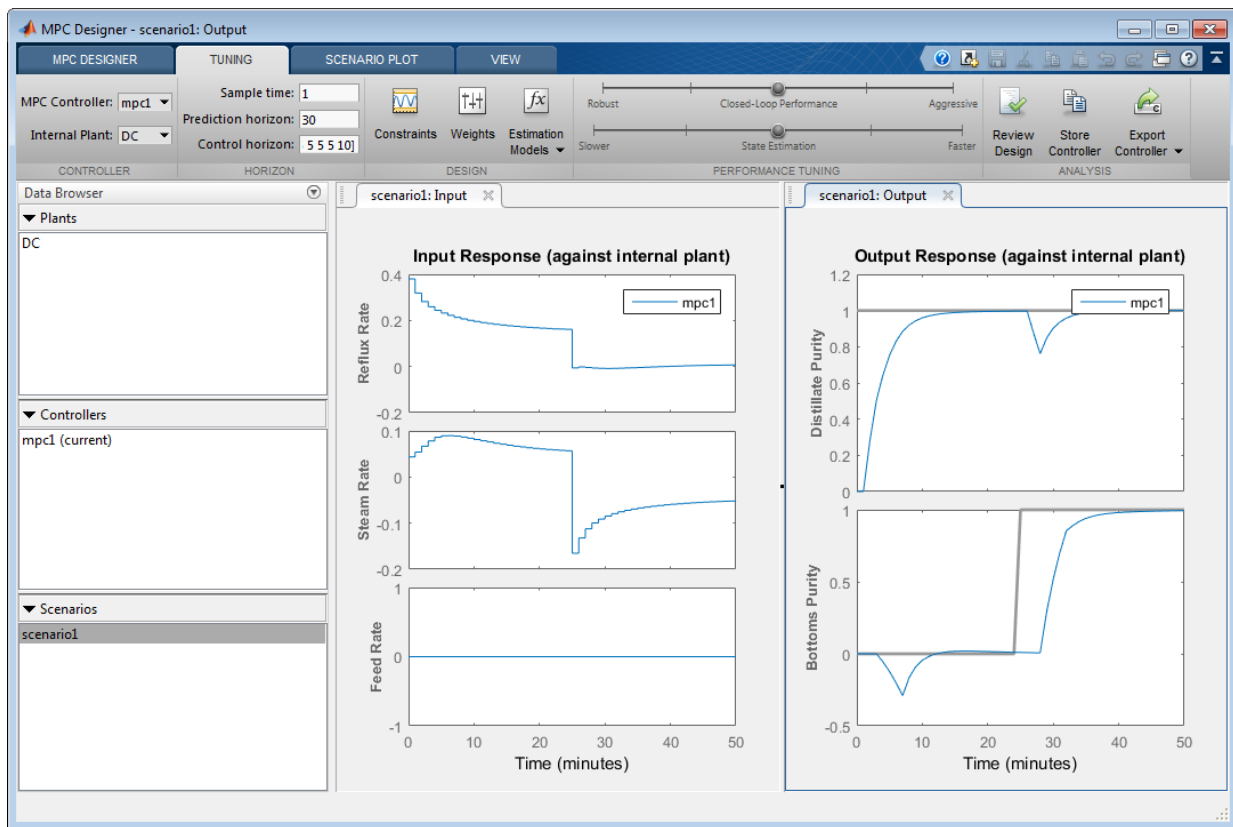
The initial manipulated variable moves are much smaller and the moves are less oscillatory. The trade-off is a slower output response, with larger interactions between the outputs.

Improve Performance By Tuning Controller Weights

Alternatively, you can produce smooth manipulated variable moves by adjusting the tuning weights of the controller.

Set the **Control horizon** back to the previous value of **5**.

In the **Performance Tuning** section, drag the **Closed-Loop Performance** slider to the left towards the **Robust** setting.
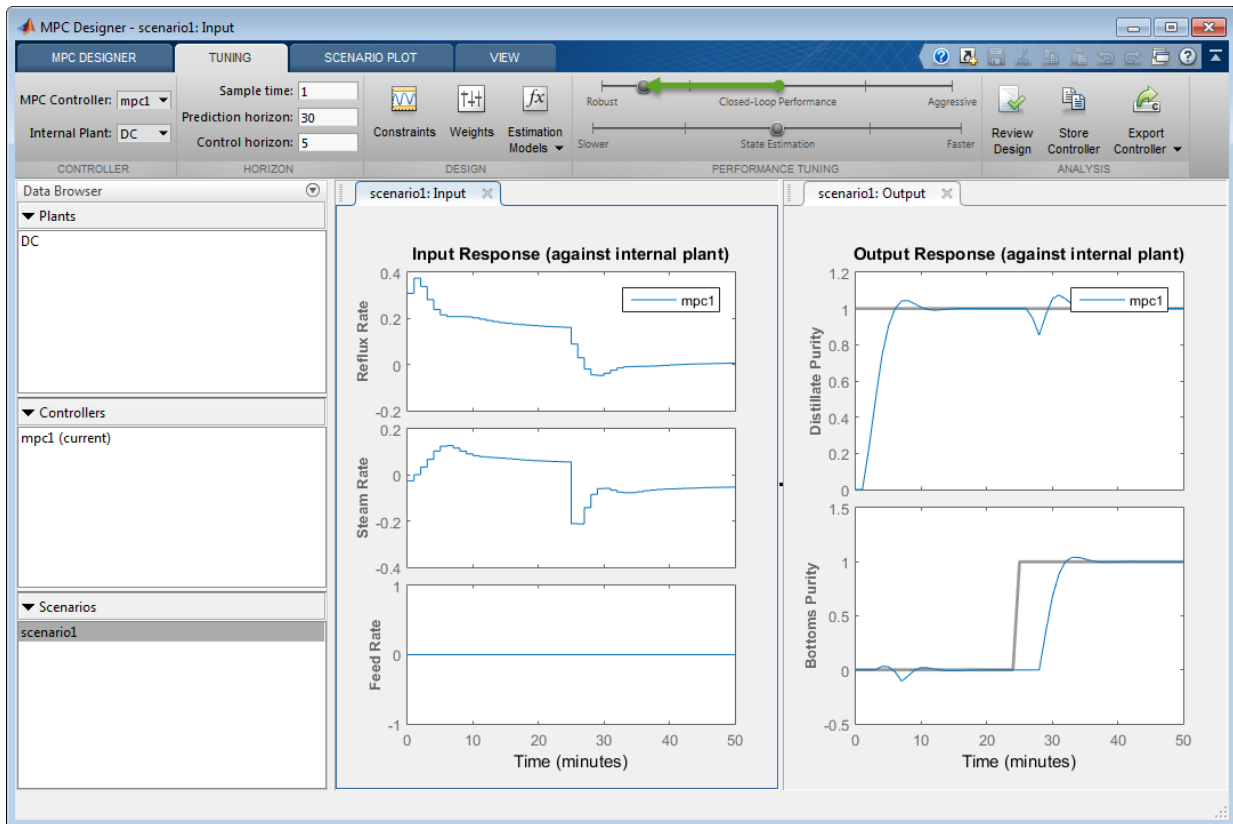
As you move the slider to the left, the manipulated variable moves become smoother and the output response becomes slower.

References

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

See Also
MPC Designer

Related Examples

- "Design Controller Using MPC Designer" on page 3-2
- "Specify Multi-Input Multi-Output (MIMO) Plants" on page 2-17

More About

- "Manipulated Variable Blocking"

Design MPC Controller for Nonsquare Plant

This topic shows how to configure an MPC controller for a nonsquare plant with unequal numbers of manipulated variables and outputs. Model Predictive Control Toolbox software supports plants with an excess of manipulated variables or plant with an excess of outputs.

More Outputs Than Manipulated Variables

When there are excess outputs, you cannot hold each at a setpoint. In this case, you have two options:

- Specify that certain outputs do not need to be held at setpoints by setting their tuning weights to zero.

  The controller does not enforce setpoints on outputs with zero weight, and the outputs are free to vary. If the plant has N_e more outputs than manipulated variables, setting N_e output weights to zero enables the controller to hold the remaining outputs at their setpoints. If any manipulated variables are constrained, one or more output responses can still exhibit steady-state error, depending on the magnitudes of reference and disturbance signals.

  Outputs with zero tuning weights can still be useful. If measured, the controller can use the outputs to help estimate the state of the plant. The outputs can also be used as performance indicators or held within an operating region defined by output constraints.

- Enforce setpoints on all outputs by specifying nonzero tuning weights for all of them.

  The controller tries to hold all outputs at their respective setpoints. However, due to the limited number of manipulated variables, all output responses exhibit some degree of steady-state error.

  You can change the error magnitudes by adjusting the relative values of the output weights. Increasing an output weight decreases the steady-state error in that output at the expense of increased error in the other outputs.

You can configure the output tuning weights at the command line by setting the `Weights.OutputVariables` property of the controller.

To configure output tuning weights in the MPC Designer app, on the **Tuning** tab, in the **Design** section, click **Weights** to open the Weights dialog box.

In the **Output Weights** section, specify the **Weight** for each output variable. For example, if your plant has two manipulated variables and three outputs, you can:

- Set one of the output weights to zero.

- Set all the weights to nonzero values. Outputs with higher weights exhibit less steady-state error.

More Manipulated Variables Than Outputs

When there are excessive manipulated variables, the default MPC controller settings allow for error-free output setpoint tracking. However, the manipulated variables values can drift. You can prevent this drift by setting manipulated variable setpoints. If there are N_e excess manipulated variables, and you hold N_e of them at target values for economic or operational reasons, the remaining manipulated variables attain the values required to eliminate output steady-state error.

To configure a manipulated variable setpoint at the command line, use the `ManipulatedVariables.Target` controller property. Then specify an input tuning weight using the controller `Weights.ManipulatedVariables` property.

To define a manipulated variable setpoint in the MPC Designer app, on the **Tuning** tab, in the **Design** section, click **Weights**.

In the Weights dialog box, in the **Input Weights** section, specify a nonzero **Weight** value for the manipulated variable.

Specify a **Target** value for the manipulated variable.

By default, the manipulated variable **Target** is nominal, which means that it tracks the nominal value specified in the controller properties.

Note: Since nominal values apply to all controllers in an MPC Designer session, changing a **Nominal Value** updates all controllers in the app. The **Target** value, however, is specific to each individual controller.

The magnitude of the manipulated variable weight indicates how much the input can deviate from its setpoint. However, there is a trade-off between manipulated variable target tracking and output reference tracking. If you want to have better output setpoint tracking performance, use a relatively small input weight. If you want the manipulated variable to stay close to its target value, increase its input weight relative to the output weight.

You can also avoid drift by constraining one or more manipulated variables to a narrow operating region using hard constraints. To define constraints in the MPC Designer app, on the **Tuning** tab, in the **Design** section, click **Constraints** to open the Constraints dialog box.

In the **Input Constraints** section, specify **Max** and **Min** constraints values.

See Also
mpc | MPC Designer

Related Examples
- "Specify Multi-Input Multi-Output (MIMO) Plants" on page 2-17
- "Setting Targets for Manipulated Variables"

More About
- "Tuning Weights"

4

Designing Controllers Using the Command Line

Design MPC Controller at the Command Line

This example shows how 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". 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);
```

Display the controller properties in the Command Window.

```
display(MPCobj)
```

```
MPC object (created on 15-Feb-2016 14:33:35):
```

```
---------------------------------------------
Sampling time:      1 (seconds)
Prediction Horizon: 10
Control Horizon:    2
Model:
          Plant: [2x2 ss]
    Disturbance: []
          Noise: []
        Nominal: [1x1 struct]

        Output disturbance model: default method (type "getoutdist(MPCobj)" for details
        Input disturbance model:  default method (type "getindist(MPCobj)" for details)

Details on 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 ]

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

State Estimation:  Default Kalman gain

Unconstrained
```

View and Modify Controller Properties

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

```
get(MPCobj)
```

```
  ManipulatedVariables (MV): [1x1 struct]
```

```
        OutputVariables (OV): [1x2 struct]
   DisturbanceVariables (DV): [1x1 struct]
                 Weights (W): [1x1 struct]
                       Model: [1x1 struct]
                          Ts: 1
                   Optimizer: [1x1 struct]
      PredictionHorizon (P): 10
             ControlHorizon: 2
                     History: [2.02e+03 2 15 14 33 35.1]
                       Notes: {}
                    UserData: []
```

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:
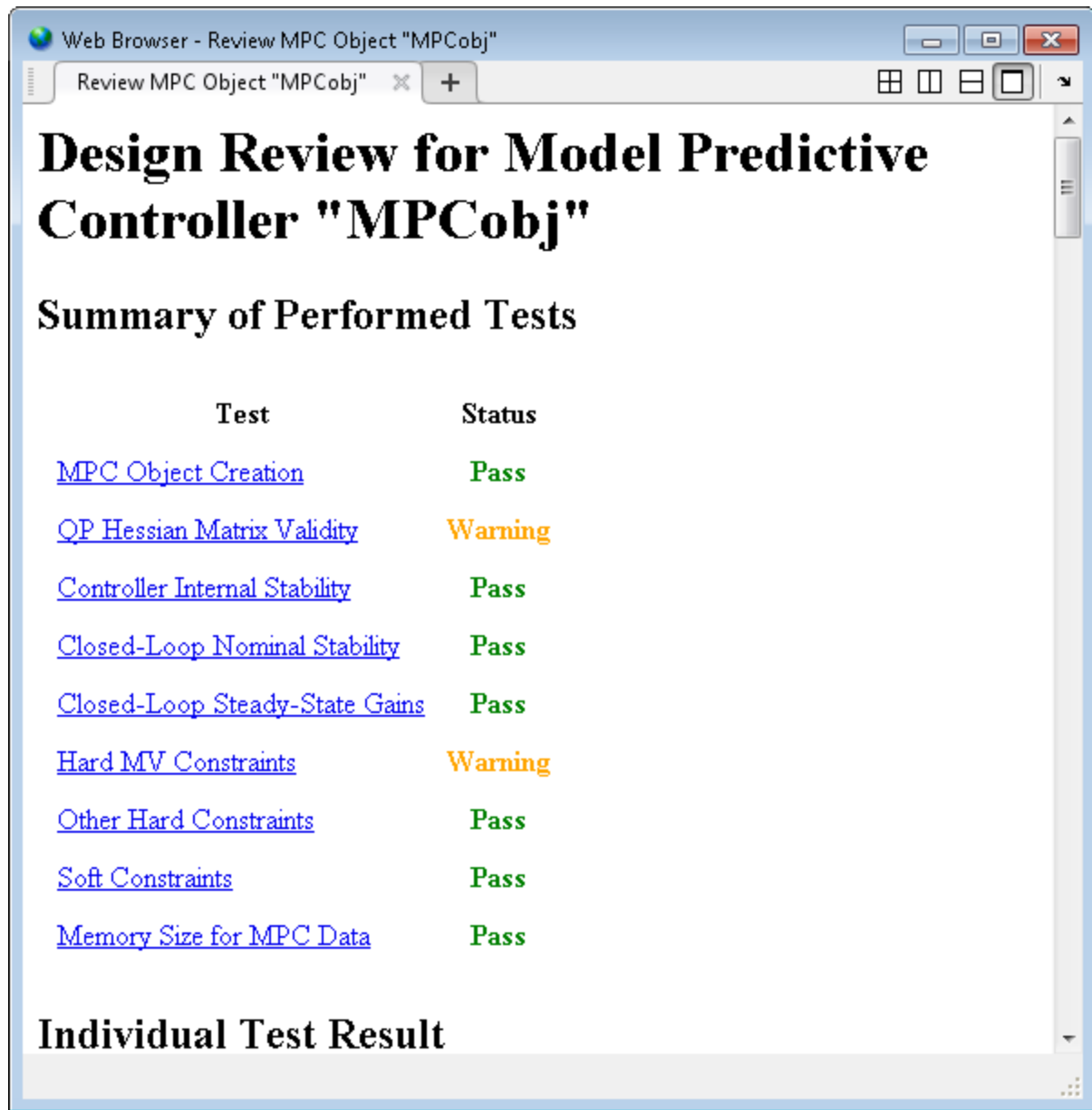
```
MPCobj.W.OutputVariables = [0.1 0; 0.2 0; 0.5 0; 1 0];
```

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)
```

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 Inputs

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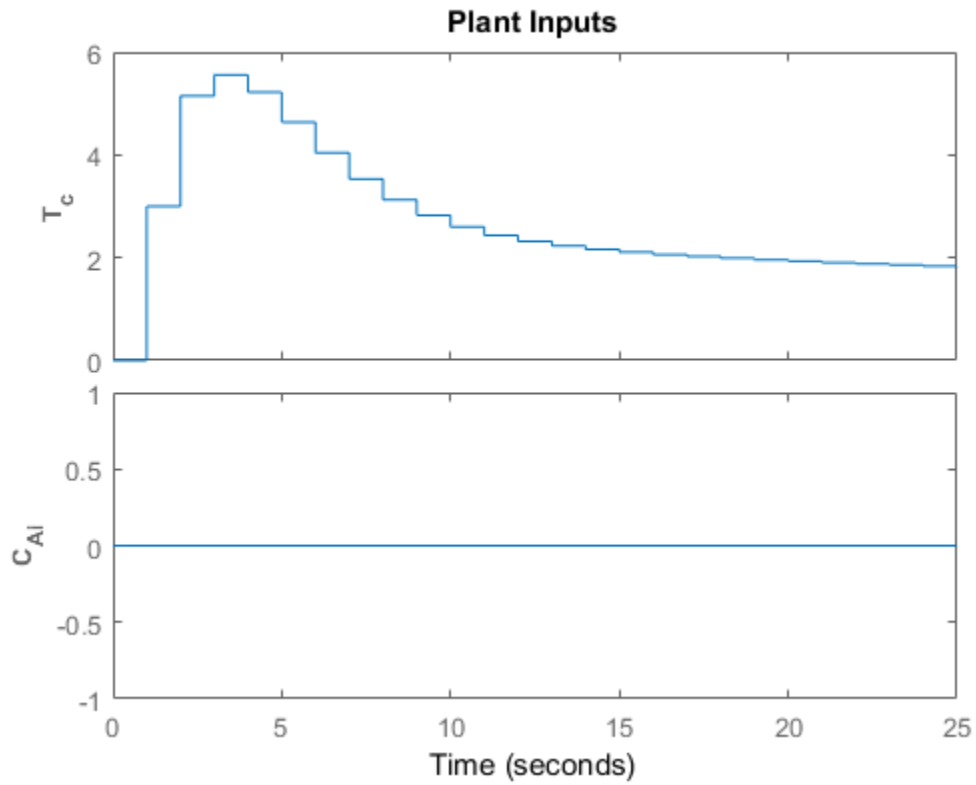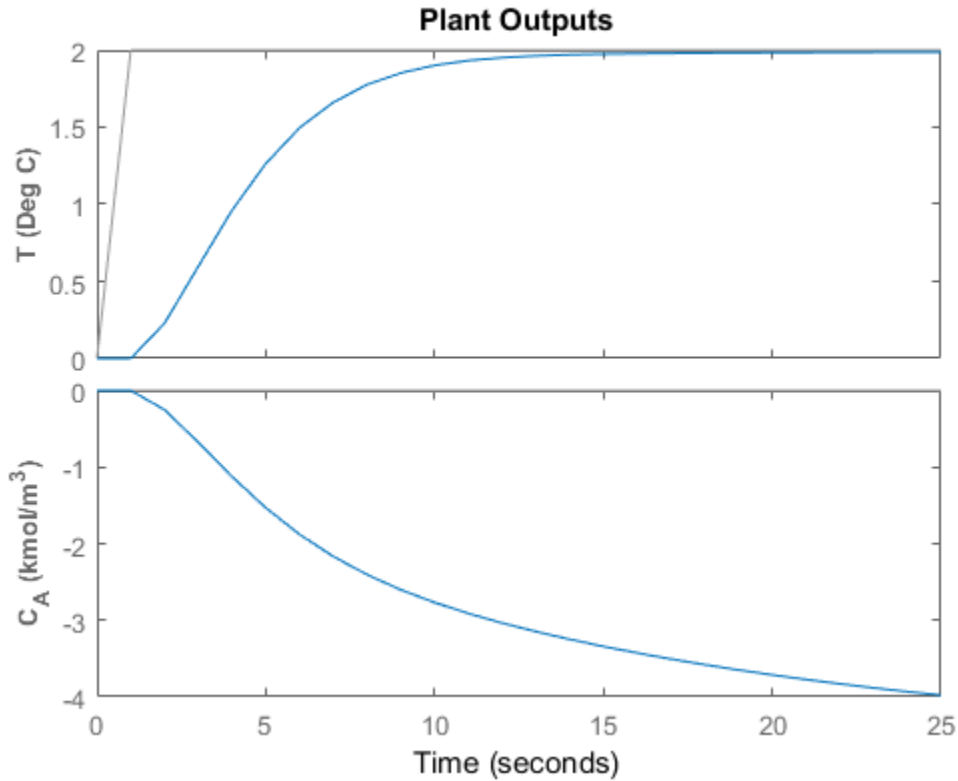
You can modify the simulation options using mpcsimopt. For example, run a simulation with the manipulated variable constraints turned off.

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

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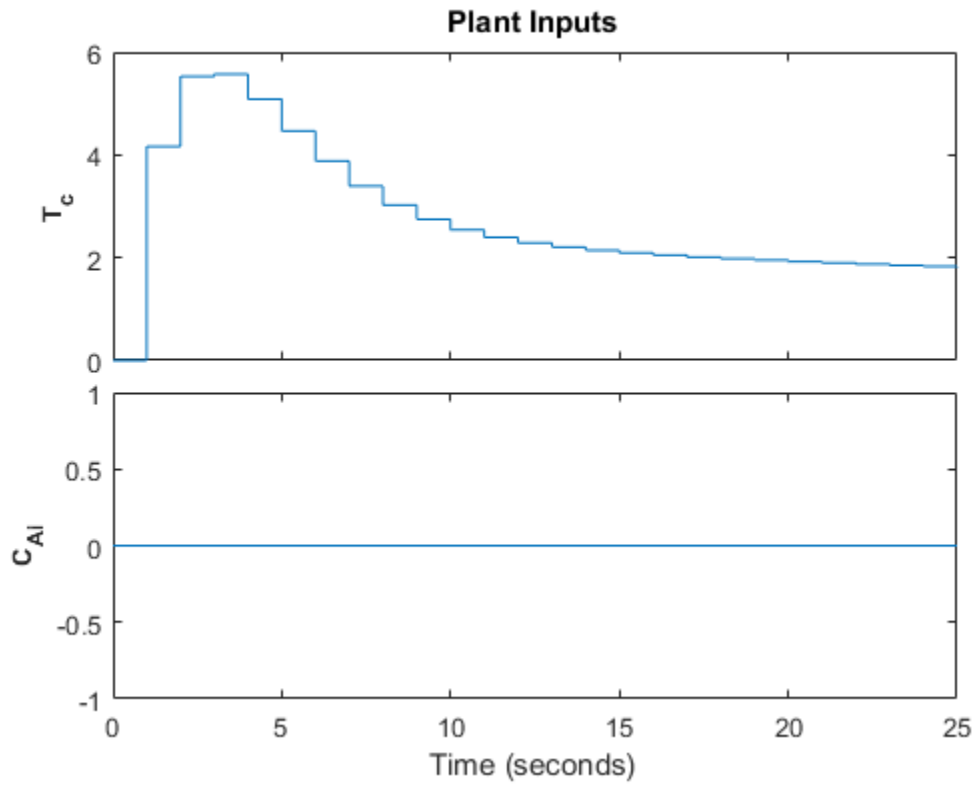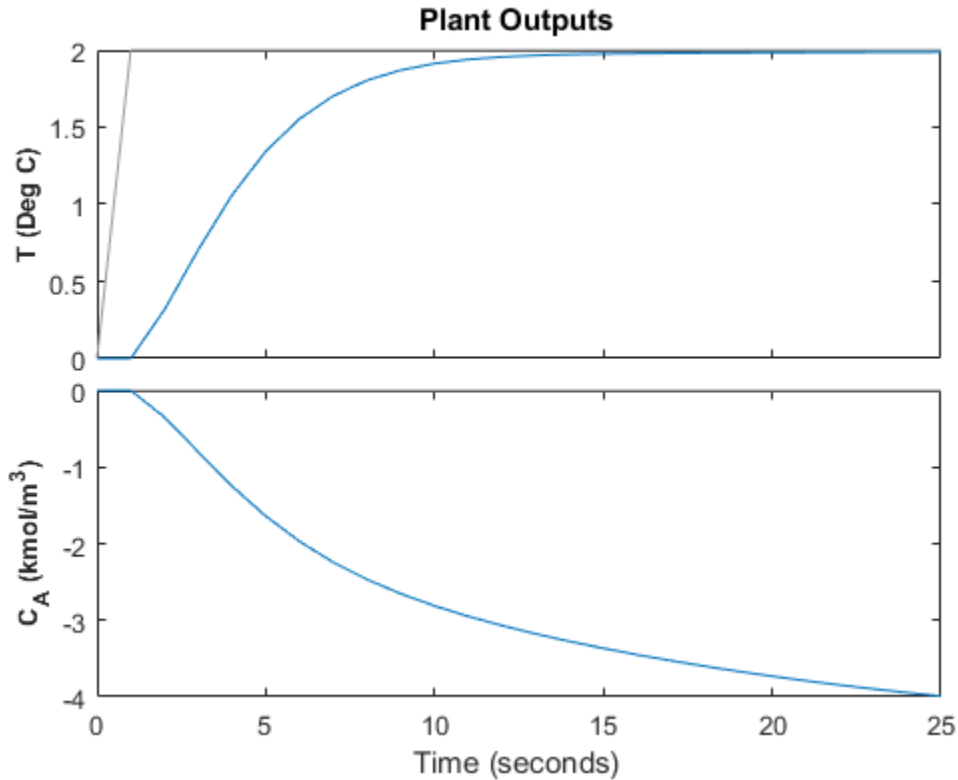
The first move of the manipulated variable now exceeds the specified 3-unit rate constraint.

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)
```

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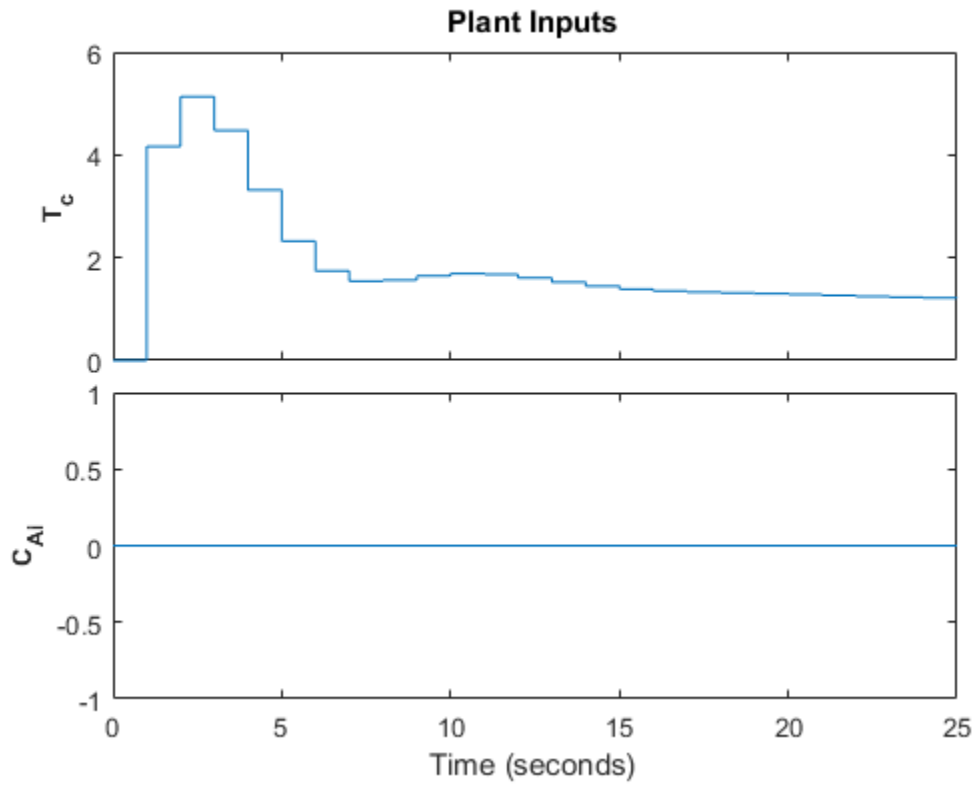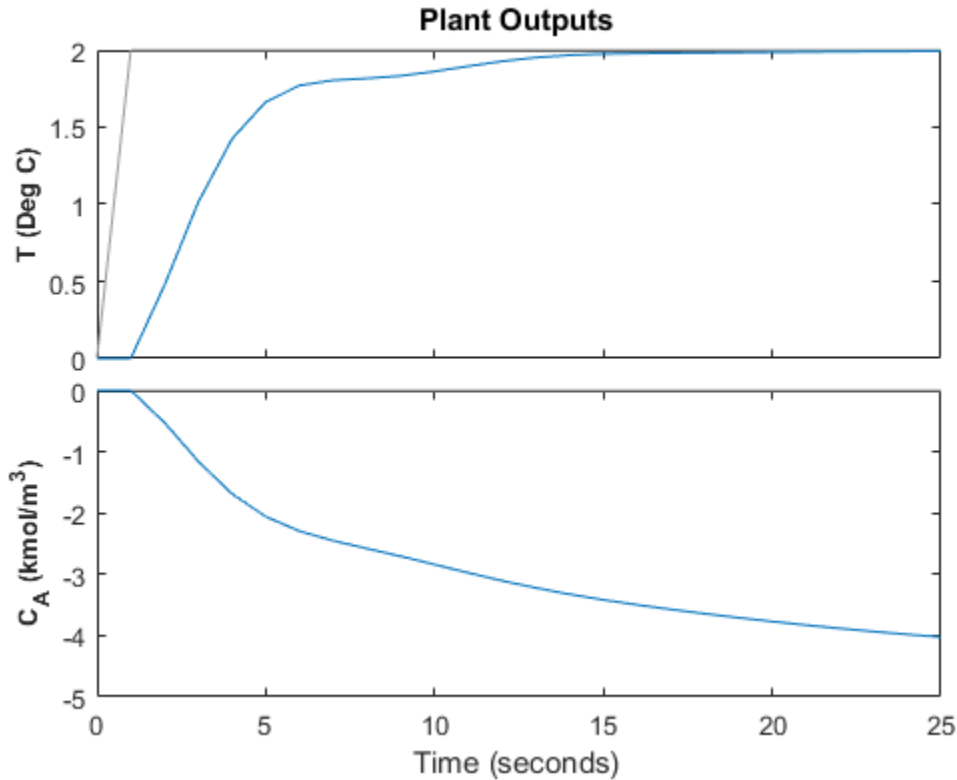
The plant/model mismatch degrades controller performance slightly. Degradation can be severe and must be tested on a case-by-case basis.

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);
```

Thie 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.

```matlab
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')
```
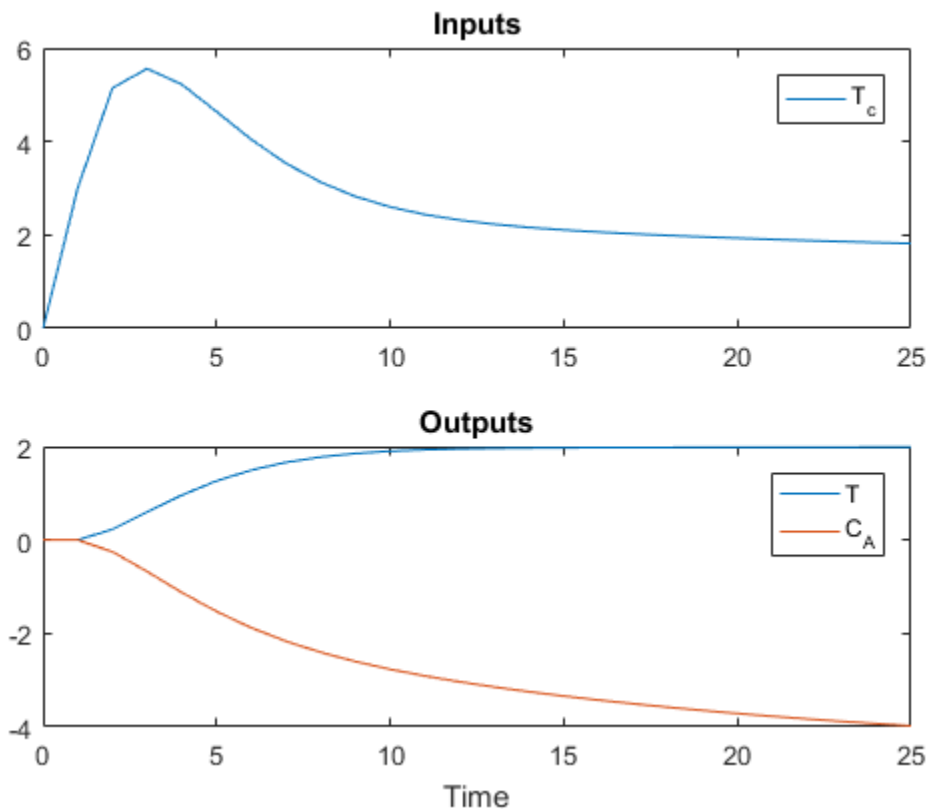
Restore the `mpcverbosity` setting.

```
mpcverbosity(old_status);
```

See Also
`mpc` | `review` | `sim`

Related Examples
- "Design Controller Using MPC Designer" on page 3-2
- "Design MPC Controller in Simulink" on page 5-2

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

Simulate Controller with Nonlinear Plant

You can use `sim` to simulate a closed-loop system consisting of a linear plant model and an MPC controller.

If your plant is a nonlinear Simulink model, you must linearize the plant (see "Linearization Using Linear Analysis Tool in Simulink Control Design" on page 2-25) and design a controller for the linear model (see "Design MPC Controller in Simulink" on page 5-2). To simulate the system, specify the controller in the MPC block parameter **MPC Controller** field and run the closed-loop Simulink model.

Alternatively, your nonlinear model might be a MEX-file, or you might want to include features unavailable in the MPC block, such as a custom state estimator. The `mpcmove` function is the Model Predictive Control Toolbox computational engine, and you can use it in such cases. The disadvantage is that you must duplicate the infrastructure that the `sim` function and the MPC block provide automatically.

The rest of this section covers the following topics:

- "Nonlinear CSTR Application" on page 4-16
- "Example Code for Successive Linearization" on page 4-17
- "CSTR Results and Discussion" on page 4-19

Nonlinear CSTR Application

The `CSTR` model described in "Linearize Simulink Models" on page 2-22 is a strongly nonlinear system. As shown in "Design MPC Controller in Simulink" on page 5-2, a controller can regulate this plant, but degrades (and might even become unstable) if the operating point changes significantly.

The objective of this example is to redefine the predictive controller at the beginning of each control interval so that its predictive model, though linear, represents the latest plant conditions as accurately as possible. This will be done by linearizing the nonlinear model repeatedly, allowing the controller to adapt as plant conditions change. See references [1] and [2] for more details on this approach.

Example Code for Successive Linearization

In the following code, the simulation begins at the nominal operating point of the CSTRmodel (concentration = 8.57) and moves to a lower point (concentration = 2) where the reaction rate is much higher. The required code is as follows:

```matlab
[sys, xp] = CSTR_INOUT([],[],[],'sizes');
up = [10 298.15 298.15];
u = up(3);
tsave = [];
usave = [];
ysave = [];
rsave = [];
Ts = 1;
t = 0;
while t < 40
    yp = xp;
    % Linearize the plant model at the current conditions
    [a,b,c,d] = linmod('CSTR_INOUT', xp, up);
    Plant = ss(a,b,c,d);
    Plant.InputGroup.ManipulatedVariables = 3;
    Plant.InputGroup.UnmeasuredDisturbances = [1 2];
    Model.Plant = Plant;

    % Set nominal conditions to the latest values
    Model.Nominal.U = [0 0 u];
    Model.Nominal.X = xp;
    Model.Nominal.Y = yp;

    dt = 0.001;

    simOptions.StartTime = num2str(t);
    simOptions.StopTime = num2str(t+dt);
    simOptions.LoadInitialState = 'on';
    simOptions.InitialState = 'xp';
    simOptions.SaveTime = 'on';
    simOptions.SaveState = 'on';
    simOptions.LoadExternalInput = 'on';
    simOptions.ExternalInput = '[t up; t+dt up]';

    simOut = sim('CSTR_INOUT',simOptions);

    T = simOut.get('tout');
    XP = simOut.get('xout');
```

```matlab
        YP = simOut.get('yout');

        Model.Nominal.DX = (1/dt)*(XP(end,:)' - xp(:));

        % Define MPC Toolbox controller for the latest model
        MPCobj = mpc(Model, Ts);
        MPCobj.W.Output = [O 1];

        % Ramp the setpoint
        r = max([8.57 - 0.25*t, 2]);

        % Compute the control action
        if t <= O
            xd = [O; O];
            x = mpcstate(MPCobj, xp, xd, [], u);
        end

        u = mpcmove(MPCobj, x, yp, [O r], []);

        % Simulate the plant for one control interval
        up(3) = u;

        simOptions.StartTime = num2str(t);
        simOptions.StopTime = num2str(t+Ts);
        simOptions.InitialState = 'xp';
        simOptions.ExternalInput = '[t up; t+Ts up]';

        simOut = sim('CSTR_INOUT',simOptions);

        T = simOut.get('tout');
        XP = simOut.get('xout');
        YP = simOut.get('yout');

        % Save results for plotting
        tsave = [tsave; T];
        ysave = [ysave; YP];
        usave = [usave; up(ones(length(T),1),:)];
        rsave = [rsave; r(ones(length(T),1),:)];

        xp = XP(end,:)';

        t = t + Ts;
    end
```

```
figure(1)
plot(tsave,[ysave(:,2) rsave])
title('Residual Concentration')
figure(2)
plot(tsave,usave(:,3))
title('Coolant Temperature')
```

CSTR Results and Discussion

The plotted results appear below. Note the following points:

- The setpoint is being ramped from the initial concentration to the desired final value (see the step-wise changes in the reactor concentration plot below). The reactor concentration tracks this ramp smoothly with some delay (see the smooth curve), and settles at the final state with negligible overshoot. The controller works equally well (and achieves the final concentration more rapidly) for a step-wise setpoint change, but it makes unrealistically rapid changes in coolant temperature (not shown).

- The final steady state requires a coolant temperature of 305.20 K (see the coolant temperature plot below). An interesting feature of this nonlinear plant is that if one starts at the initial steady state (coolant temperature = 298.15 K), stepping the coolant temperature to 305.20 and holding will not achieve the desired final concentration of 2. In fact, under this simple strategy the reactor concentration stabilizes at a final value of 7.88, far from the desired value. A successful controller must increase the reactor temperature until the reaction "takes off," after which it must reduce the coolant temperature to handle the increased heat load. The relinearization approach provides such a controller (see following plots).
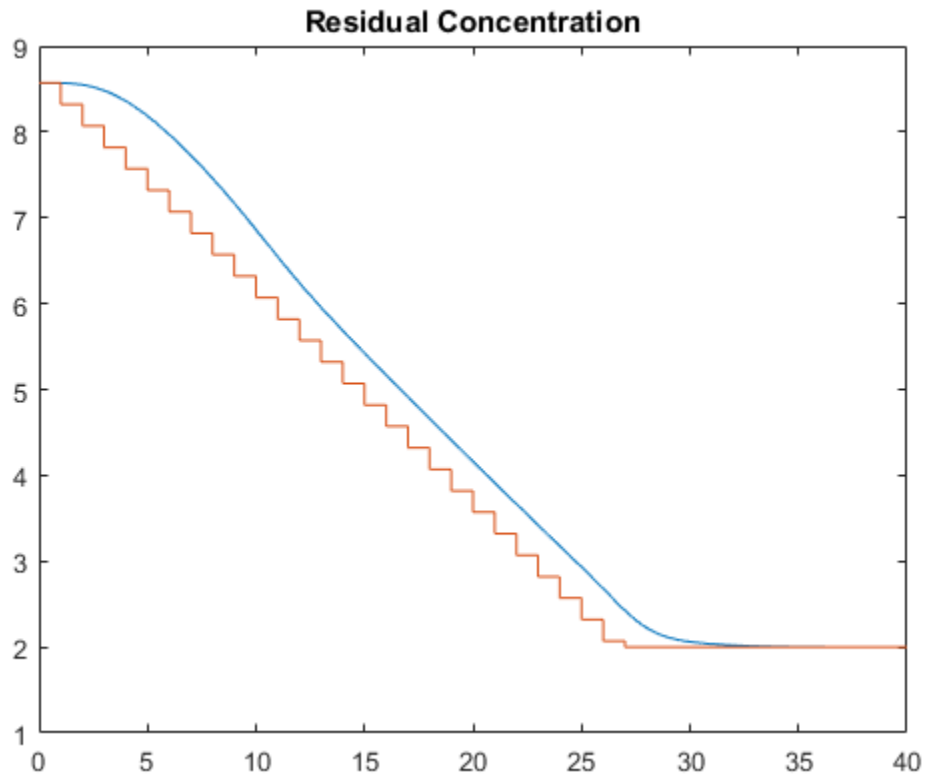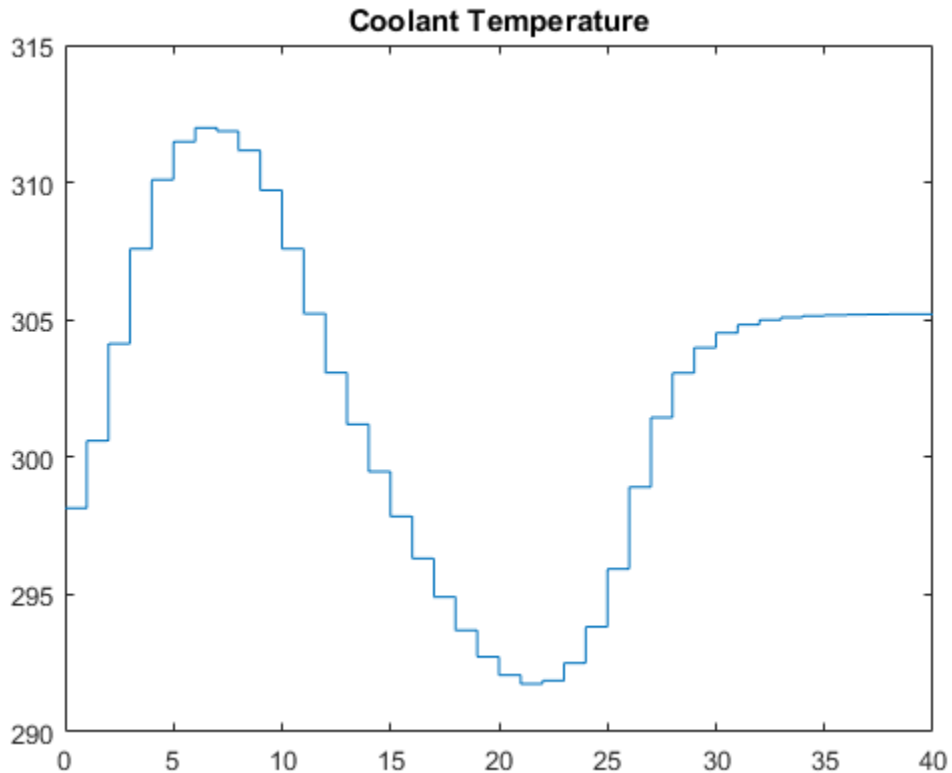
- Function `linearize` relinearizes the plant as its state evolves. This function was discussed previously in "Linearization Using MATLAB Code" on page 2-22.

- The code also resets the linear model's nominal conditions to the latest values. Note, however, that the first two input signals, which are unmeasured disturbances in the controller design, always have nominal zero values. As they are unmeasured, the controller cannot be informed of the true values. A non-zero value would cause an error.

- Function `mpc` defines a new controller based on the relinearized plant model. The output weight tuning ignores the temperature measurement, focusing only on the concentration.

- At $t = 0$, the `mpcstate` function initializes the controller's extended state vector, `x`, which is an *mpcstate object*. Thereafter, the `mpcmove` function updates it

automatically using the controller's default state estimator. It would also be possible to use an Extended Kalman Filter (EKF) as described in [1] and [2], in which case the EKF would reset the `mpcstate` input variables at each step.

- The `mpcmove` function uses the latest controller definition and state, the measured plant outputs, and the setpoints to calculate the new coolant temperature at each step.

- The Simulink `sim` function simulates the nonlinear plant from the beginning to the end of the control interval. Note that the final condition from the previous step is being used as the initial plant state, and that the plant inputs are being held constant during each interval.

Remember that a conventional feedback controller or a fixed Model Predictive Control Toolbox controller tuned to operate at the initial condition would become unstable as the plant moves to the final condition. Periodic model updating overcomes this problem automatically and provides excellent control under all conditions.

References

[1] Lee, J. H. and N. L. Ricker, "Extended Kalman Filter Based Nonlinear Model Predictive Control," *Ind. Eng. Chem. Res.*, Vol. 33, No. 6, pp. 1530–1541 (1994).

[2] Ricker, N. L., and J. H. Lee "Nonlinear Model Predictive Control of the Tennessee Eastman Challenge Process," *Computers & Chemical Engineering*, Vol. 19, No. 9, pp. 961–981 (1995).

Control Based On Multiple Plant Models

The "Nonlinear CSTR Application" on page 4-16 shows how updates to the prediction model can improve MPC performance. In that case the model is nonlinear and you can obtain frequent updates by linearization.

A more common situation is that you have several linear plant models, each of which applies at a particular operating condition and you can design a controller based on each linear model. If the models cover the entire operating region and you can define a criterion by which you switch from one to another as operating conditions change, the controller set should be able to provide better performance than any individual controller.

The Model Predictive Control Toolbox includes a Simulink block that performs this function. It is the *Multiple MPC Controllers* block. The rest of this section is an illustrative example organized as follows:

- "A Two-Model Plant" on page 4-23
- "Designing the Two Controllers" on page 4-25
- "Simulating Controller Performance" on page 4-26

A Two-Model Plant

Note Switching MPC Controllers with Multiple MPC Controllers Block provides an animated version of the plant described below.

Animation of the Multi-Model Example is a stop-action snapshot of the subject plant. It consists of two masses, M_1 and M_2. A spring connects M_1 to a rigid wall and pulls it to the right. An applied force, shown as a red arrow in Animation of the Multi-Model Example, opposes this spring, pulling M_1 to the left.

When the two masses are detached, as in Animation of the Multi-Model Example, mass M_2 is uncontrollable and responds only to the spring pulling it to the left.

If the two masses collide, however, they stick together (the collision is completely inelastic) until a change in the applied force separates them.

The control objective is to move M_1 in response to a command signal. The blue triangle in Animation of the Multi-Model Example represents the desired location. At the instant shown, the desired location is –5.
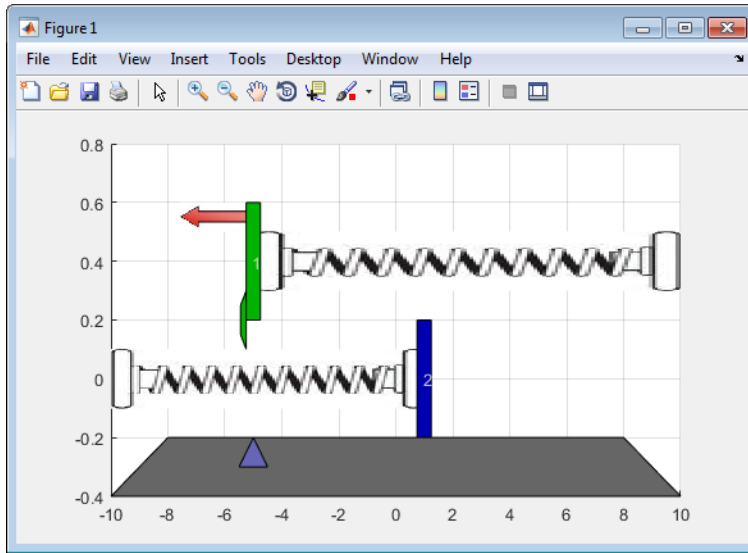
4-23

Animation of the Multi-Model Example

In order to achieve its objective, the controller can adjust the applied force magnitude (the length of the red arrow). It receives continuous feedback on the M_1 location. There is also a contact sensor to signal collisions. The M_2 location is unmeasured.

If M_1 were isolated, this would be a routine control problem. The challenge is that the relationship between the applied force and the M_1 movement changes dramatically when M_2 attaches to M_1.

The following code defines the model. First define the system parameters as follows:

```
%% Model Parameters
M1 = 1;        % mass
M2 = 5;        % mass
k1 = 1;        % spring constant
k2 = 0.1;      % spring constant
b1 = 0.3;      % friction coefficient
b2 = 0.8;      % friction coefficient
yeq1 = 10;     % wall mount position
yeq2 = -10;    % wall mount position
```

Next define a model of M_1 when the masses are separated. Its states are the M_1 position and velocity. Its inputs are the applied force, which will be the controller's manipulated

variable, and a spring constant calibration signal, which is a measured disturbance input.

```
A1 = [0 1;-k1/M1 -b1/M1];
B1 = [0 0;-1/M1 k1*yeq1/M1];
C1 = [1 0];
D1 = [0 0];
sys1 = ss(A1,B1,C1,D1);
sys1 = setmpcsignals(sys1,'MV',1,'MD',2);
```

The `setmpcsignals` command specifies the input type for the two inputs.

We need another model (with the same input/output structure) to predict movement when the two masses are joined, as follows:

```
A2 = [0 1;-(k1+k2)/(M1+M2) -(b1+b2)/(M1+M2)];
B2 = [0 0;-1/(M1+M2) (k1*yeq1+k2*yeq2)/(M1+M2)];
C2 = [1 0];
D2 = [0 0];
sys2 = ss(A2,B2,C2,D2);
sys2 = setmpcsignals(sys2,'MV',1,'MD',2);
```

Designing the Two Controllers

Next we define controllers for each case. Both use a sample time of `0.2`, a prediction horizon of $P = 20$, a control horizon of $M = 1$, and the default values for all other controller design parameters. The only difference in the controllers is the prediction model.

```
Ts = 0.2;       % sampling time
p = 20;         % prediction horizon
m = 1;          % control horizon
MPC1 = mpc(sys1,Ts,p,m); % Controller for M1 detached from M2
MPC2 = mpc(sys2,Ts,p,m); % Controller for M1 connected to M2
```

The applied force also has the same constraints in each case. Its lower bound is zero (it can't reverse direction), and its maximum rate of change is 1000 per second (increasing or decreasing).

```
MPC1.MV = struct('Min',0,'RateMin',-1e3,'RateMax',1e3);
MPC2.MV = struct('Min',0,'RateMin',-1e3,'RateMax',1e3);
```

Simulating Controller Performance

Block Diagram of the Two-Model Example shows the Simulink block diagram for this example. The upper portion simulates the movement of the two masses, plots the signals as a function of time, and animates the example.
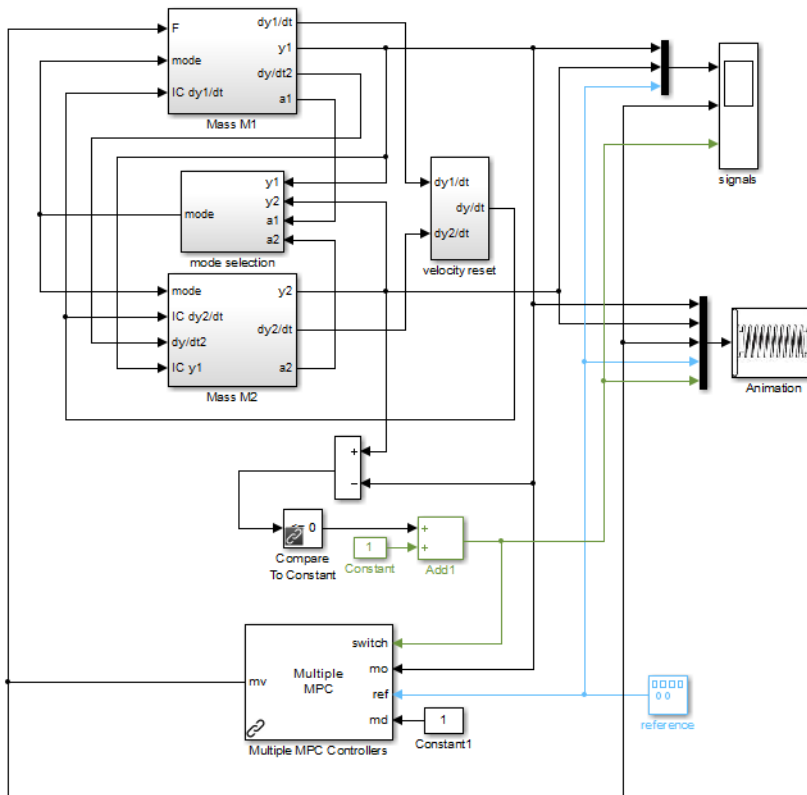
Block Diagram of the Two-Model Example

The lower part contains three key elements:

- A pulse generator that supplies the desired M_1 position (the controller reference signal). Its output is a square wave varying between −5 and 5 with a frequency of 0.015 per second.

- A simulation of a contact sensor. When the two masses have the same position, the `Compare to Constant` block evaluates to `true`, and the `Add1` block converts this to a 2. Otherwise, the `Add1` output is 1.

- The Multiple MPC Controller block. It has four inputs. The measured output (`mo`), reference (`ref`), and measured disturbance (`md`) inputs are as for a standard MPC Controller block. The distinctive feature is the `switch` input.

The figure below shows the Multiple MPC Controller block mask for this example (obtained by double clicking on the controller block).
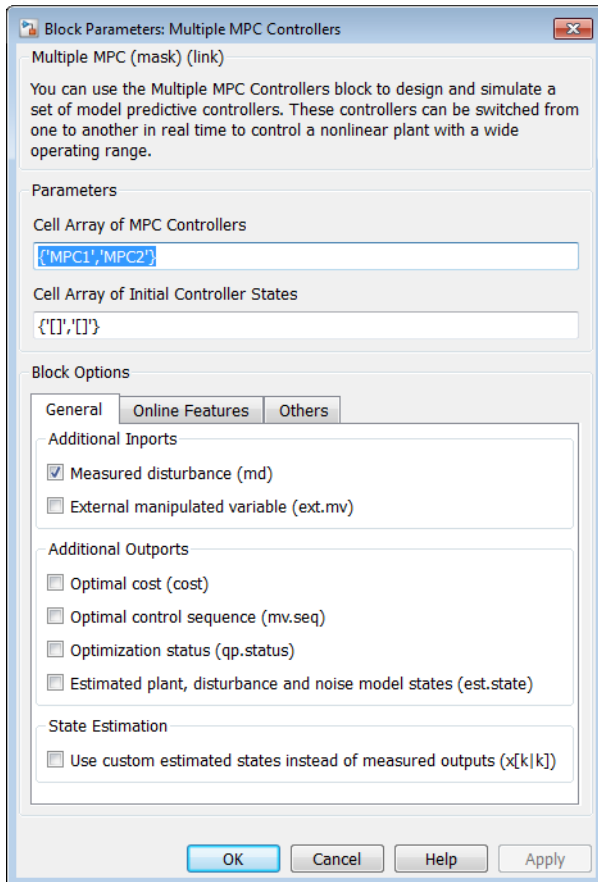
When the `switch` input is 1 the block automatically activates the first controller listed (`MPC1`), which is appropriate when the masses are separated. When the `switch` input is 2 the block automatically enables the second controller (`MPC2`).

The following code simulates the controller performance

```
Tstop = 100;        % Simulation time
y1initial = 0;      % Initial M1 and M2 Positions
y2initial = 10;
open('mpc_switching')
sim('mpc_switching',Tstop)
```

The figure below shows the `signals` scope output.
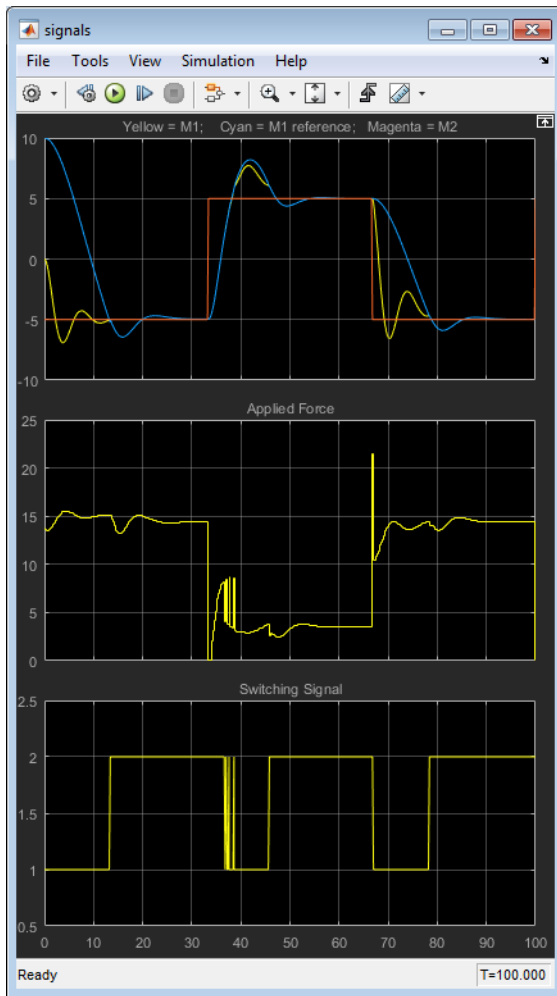
In the upper plot, the cyan curve is the desired position. It starts at −5. The M_1 position (yellow) starts at 0 and under the control of MPC1, M_1 moves rapidly toward the desired position. M_2 (magenta) starts at 10 and begins moving in the same direction. At about t = 13 seconds, M_2 collides with M_1. The switching signal (lower plot) changes at this instant from 1 to 2, so controller MPC2 has taken over.

The collision moves M_1 away from its desired position and M_2 remains joined to M_1. Controller MPC2 adjusts the applied force (middle plot) so M_1 quickly returns to the desired position.

When the desired position changes step-wise to 5, the two masses separate briefly (with appropriate switching to MPC1) but for the most part move together and settle rapidly at the desired position. The transition back to –5 is equally well behaved.

Now suppose we force MPC2 to operate under all conditions. The figure below shows the result. When the masses are separated, as at the start, MPC2 applies excessive force and then over-compensates, resulting in oscillatory behavior. Once the masses join, the movement smooths out, as would be expected.

The oscillations are especially severe in the last transition. The masses collide frequently and M_1 never reaches the desired position.
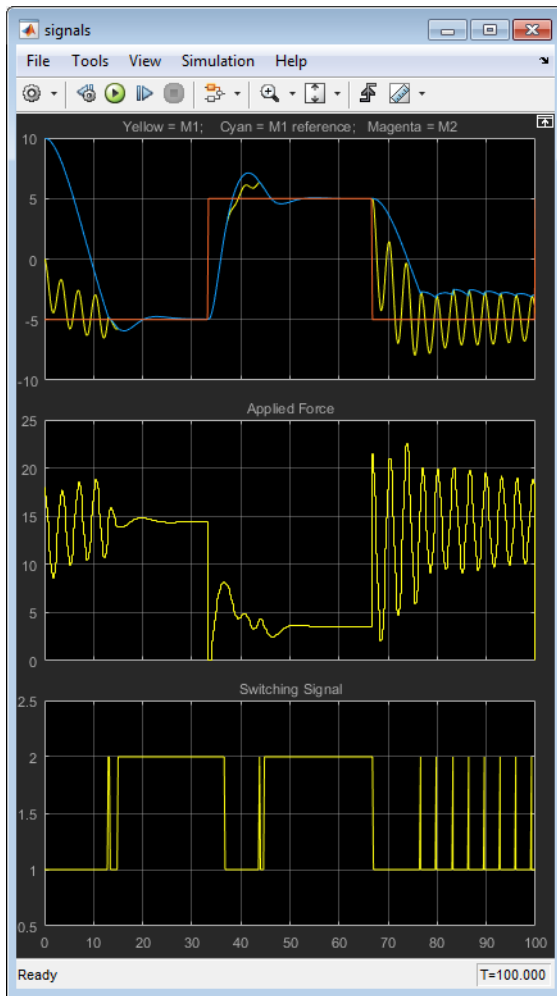
If we put `MPC1` in charge exclusively, we instead see sluggish movements that fail to settle at the desired position before the next transition occurs (not shown but you can run `mpcswitching`).

In this case, at least, two controllers are better than one.

Compute Steady-State Gain

This example shows how to analyze a model predictive controller using cloffset. This function computes the closed-loop, steady-state gain for each output when a sustained, 1-unit disturbance is added to each output. It assumes that no constraints are active.

Define a state-space plant model.

```
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.InputGroup.MV = 1;
CSTR.InputGroup.UD = 2;
```

Create an MPC controller for the defined plant.

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

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->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.
   for output(s) y1 and zero weight for output(s) y2
```

Specify tuning weights for the measured output signals.

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

Compute the closed-loop, steady-state gain for this controller.

```
DCgain = cloffset(MPCobj)
```

```
-->Converting model to discrete time.
-->The "Model.Disturbance" property of "mpc" object is empty:
   Assuming unmeasured input disturbance #2 is integrated white noise.
-->Assuming output disturbance added to measured output channel #1 is integrated white
   Assuming no disturbance added to measured output channel #2.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac

DCgain =
```

```
0.0000   -0.0000
2.3272    1.0000
```

DCgain(i,j) represents the gain from the sustained, 1-unit disturbance on output j to measured output i.

The second column of DCgain shows that the controller does not react to a disturbance applied to the second output. This disturbance is ignored because the tuning weight for this channel is 0.

Since the tuning weight for the first output is nonzero, the controller reacts when a disturbance is applied to this output, removing the effect of the disturbance (DCgain(1,1) = 0). However, since the tuning weight for the second output is 0, this controller reaction introduces a gain for output 2 (DCgain(2,1) = 2.3272).

See Also
cloffset | mpc

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

Extract Controller

This example shows how to obtain an LTI representation of an unconstrained MPC controller using ss. You can use this to analyze the frequency response and performance of the controller.

Define a plant model. For this example, use the CSTR model described in "Design Controller Using MPC Designer".

```
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.InputGroup.MV = 1;
CSTR.InputGroup.UD = 2;
CSTR.OutputGroup.MO = 1;
CSTR.OutputGroup.UO = 2;
```

Create an MPC controller for the defined plant using the same sample time, prediction horizon, and tuning weights described in "Design MPC Controller at the Command Line".

```
MPCobj = mpc(CSTR,1,15);
MPCobj.W.ManipulatedVariablesRate = 0.3;
MPCobj.W.OutputVariables = [1 0];
```

```
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->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.
   for output(s) y1 and zero weight for output(s) y2
```

Extract the LTI state-space representation of the controller.

```
MPCss = ss(MPCobj);
```

```
-->Converting model to discrete time.
-->The "Model.Disturbance" property of "mpc" object is empty:
   Assuming unmeasured input disturbance #2 is integrated white noise.
   Assuming no disturbance added to measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

Convert the original CSTR model to discrete form using the same sample time as the MPC controller.

```
CSTRd = c2d(CSTR,MPCss.Ts);
```

Create an LTI model of the closed-loop system using feedback. Use the manipulated variable and measured output for feedback, indicating a positive feedback loop. Using negative feedback would lead to an unstable closed-loop system, because the MPC controller is designed to use positive feedback.

```
CLsys = feedback(CSTRd,MPCss,1,1,1);
```

You can then analyze the resulting feedback system. For example, verify that all closed-loop poles are within the unit circle.
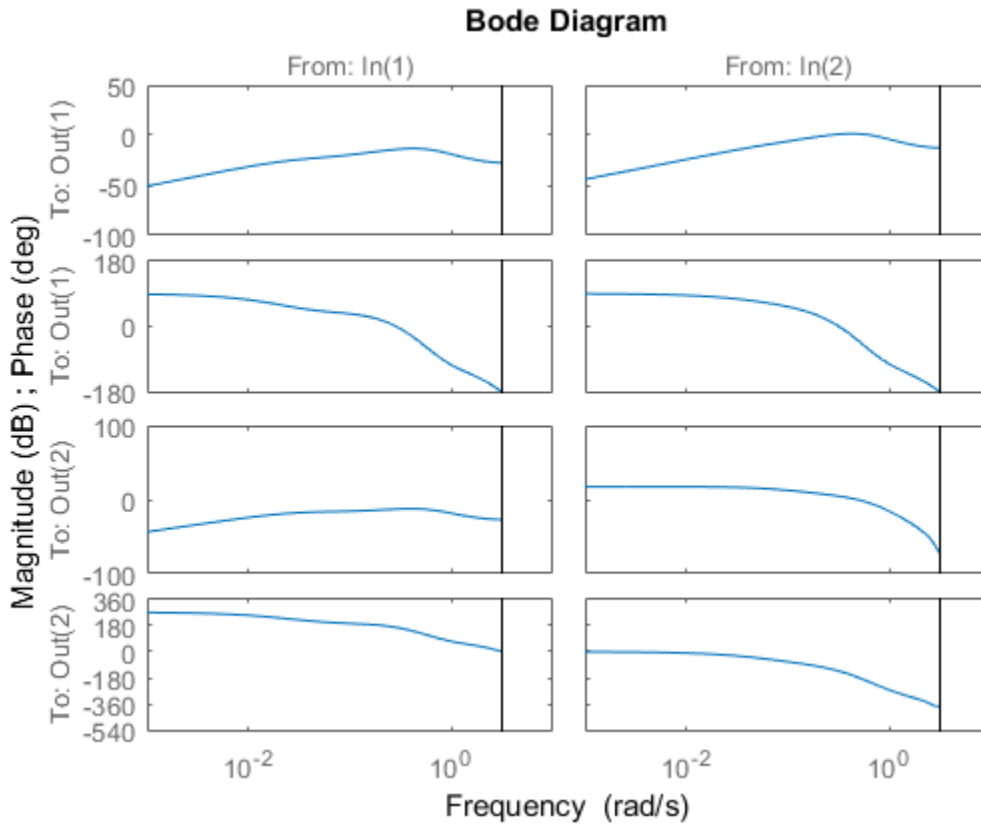
```
poles = eig(CLsys)


poles =

   0.5513 + 0.2700i
   0.5513 - 0.2700i
   0.6131 + 0.1110i
   0.6131 - 0.1110i
   0.9738 + 0.0000i
   0.9359 + 0.0000i
```

You can also view the system frequency response.

```
bode(CLsys)
```

Bode Diagram

See Also

`feedback | mpc | ss`

Related Examples

- "Design MPC Controller at the Command Line" on page 4-2

Signal Previewing

By default, a model predictive controller assumes that the current reference and measured disturbance signals remain constant during the controller prediction horizon. By doing so, the controller emulates a conventional feedback controller.

However, as shown in "Optimization Problem", these signals can vary within the prediction horizon. If your application allows you to anticipate trends in such signals, you can use signal previewing with an MPC controller to improve reference tracking, measured disturbance rejection, or both.

The following Model Predictive Control Toolbox commands provide previewing options:

- `sim`
- `mpcmove`
- `mpcmoveAdaptive`

For Simulink, the following blocks support previewing:

- `MPC Controller`
- `Adaptive MPC Controller`
- `Multiple MPC Controllers`

Previewing for explicit MPC controllers will be supported in a future release.

In the MPC Designer app, you can specify whether simulation scenarios use previewing. When editing a scenario in the Simulation Scenario dialog box, select the **Preview references** or **Preview measured disturbances** options.
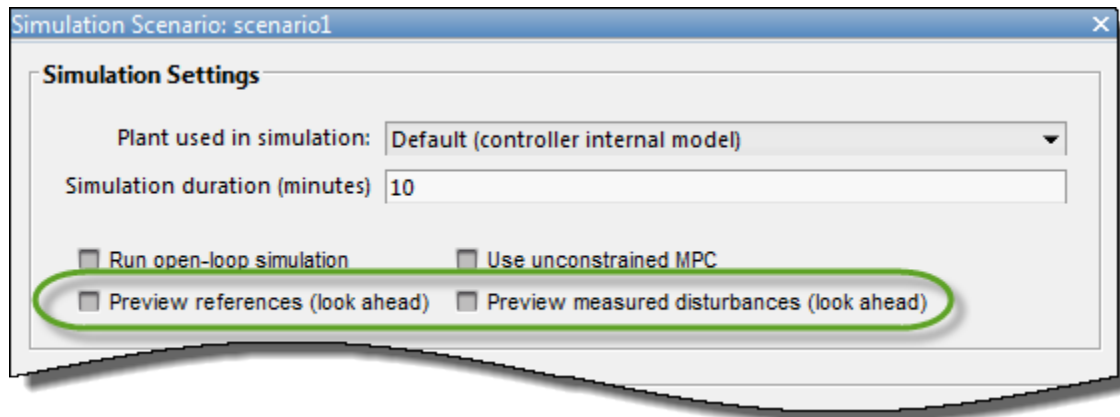
Related Examples

- Improving Control Performance with Look-Ahead (Previewing)

More About

- "Run-Time Constraint Updating" on page 4-39

Run-Time Constraint Updating

Constraint bounds can change during controller operation. The `mpcmove`, `mpcmoveAdaptive`, and `mpcmoveExplicit` commands allow for this possibility. In Simulink, all the MPC Controller blocks allow for it as well. Both of these simulation approaches give you the option to update all specified bounds at each control interval. You cannot constrain a variable for which the corresponding controller object property is unbounded. If you already defined time-varying constraints in the `mpc` controller object, the new value is applied to the first finite value in the prediction horizon. All other prediction horizon values adjust to maintain the same profile, that is they update by the same amount.

If your controller uses custom constraints on linear combinations of inputs and outputs, you can update these constraints at each simulation iteration by calling `setconstraint` before `mpcmove`. However, updating the custom constraint matrices at run time is not supported in Simulink. To deploy an MPC controller with run-time updating of custom constraints, use MATLAB Compiler™ to generate the executable code, and deploy it using the MATLAB Runtime. In this case, the controller sample time must be large, since run-time MPC regeneration is slow.

See Also
`mpcmove` | `mpcmoveAdaptive` | `mpcmoveExplicit` | `setconstraint`

Related Examples
- Varying Input and Output Constraints

More About
- "Run-Time Weight Tuning" on page 4-40
- "Constraints on Linear Combinations of Inputs and Outputs"

Run-Time Weight Tuning

There are two ways to perform tuning experiments using Model Predictive Control Toolbox software:

- Modify your controller object off line (by changing weights, etc.) and then test the modified object.
- Change tuning weights as the controller operates.

This topic describes the second approach.

You can adjust the following tuning weights as the controller operates (see "Tuning Weights"):

- Plant output variable (OV) reference tracking, w^y.
- Manipulated variable (MV) reference tracking, w^u.
- MV increment suppression, $w^{\Delta u}$.
- Global constraint softening, ρ_ϵ.

In each case, the weight applies to the entire prediction horizon. If you are using time-varying weights, you must use offline modification of the weight and then test the modified controller.

In Simulink, the following blocks support online (run-time) tuning:

- `MPC Controller`
- `Adaptive MPC Controller`
- `Multiple MPC Controllers`. In this case, the tuning signals apply to the active controller object, which might switch as the control system operates. If the objects in your set employ different weights, you should tune them off line.

The `Explicit MPC Controller` block cannot support online tuning because a weight change requires a complete revision of the explicit MPC control law, which is computationally intensive.

For command-line testing, the `mpcmove` and `mpcmoveAdaptive` commands include options to mimic the online tuning behavior available in Simulink.

Related Examples

- Tuning Controller Weights

More About

- • "Signal Previewing" on page 4-37

Designing and Testing Controllers in Simulink

- "Design MPC Controller in Simulink" on page 5-2
- "Test an Existing Controller" on page 5-23
- "Schedule Controllers at Multiple Operating Points" on page 5-27

The Model Predictive Control Toolbox provides a controller block library for use in Simulink.

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 the MPC Designer app.

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

CSTR Model

A CSTR is a jacketed nonadiabatic tank reactor commonly used in the process industry.
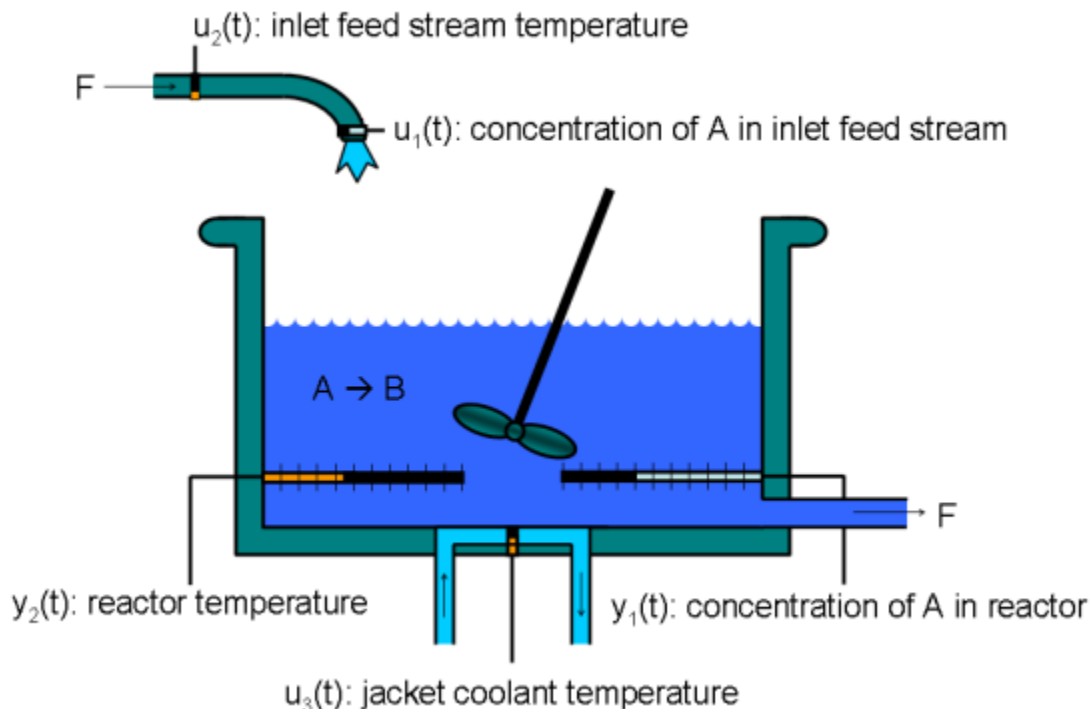
An inlet stream of reagent *A* feeds into the tank at a constant rate. A first-order, irreversible, exothermic reaction takes place to produce the product stream, which exits the reactor at the same rate as the input stream.

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');
```
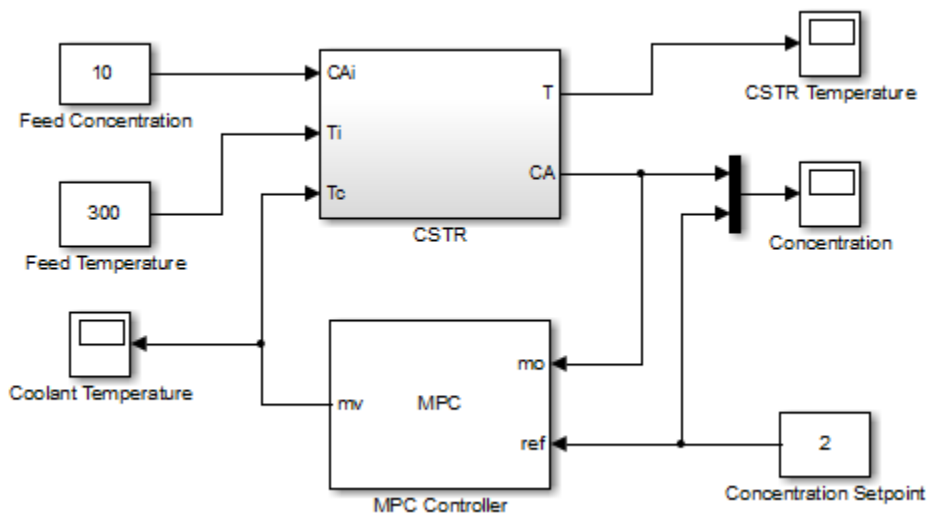
Connect Measured Disturbance To `MPC Controller` Block

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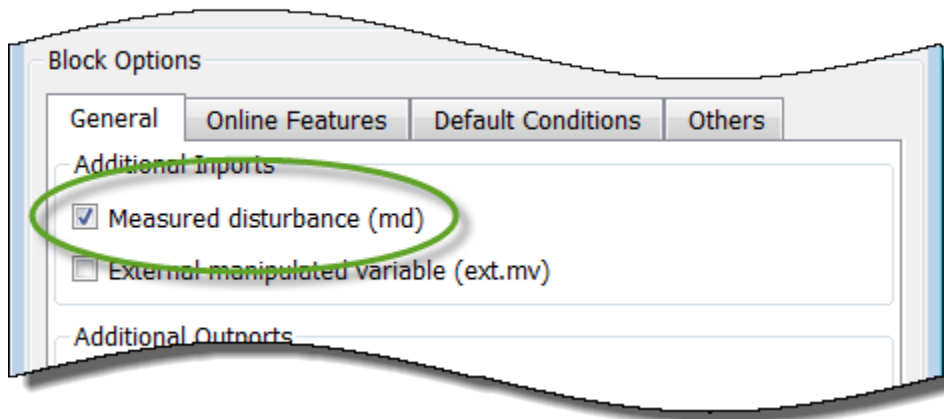
Click **Apply** to add the `md` inport to the controller block.

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.

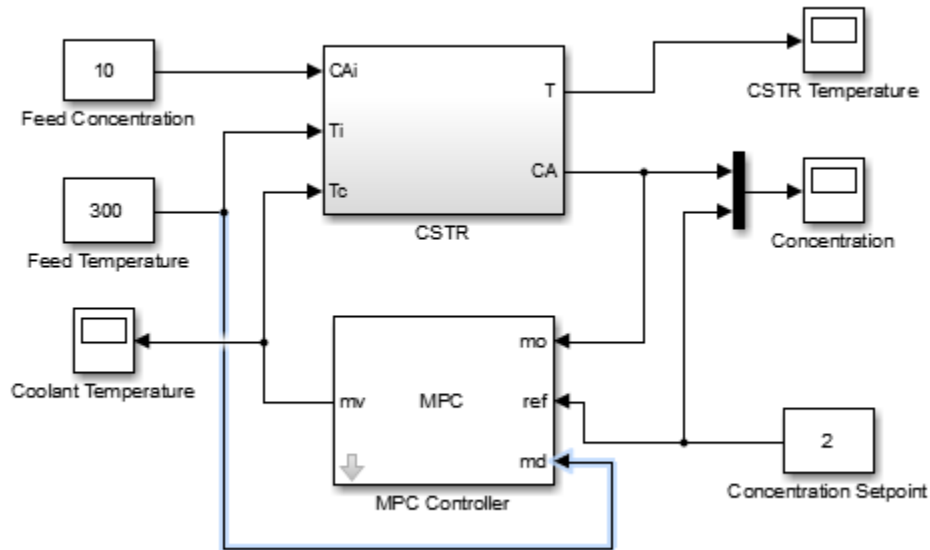### Define MPC Structure

In the MPC Designer app, 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:
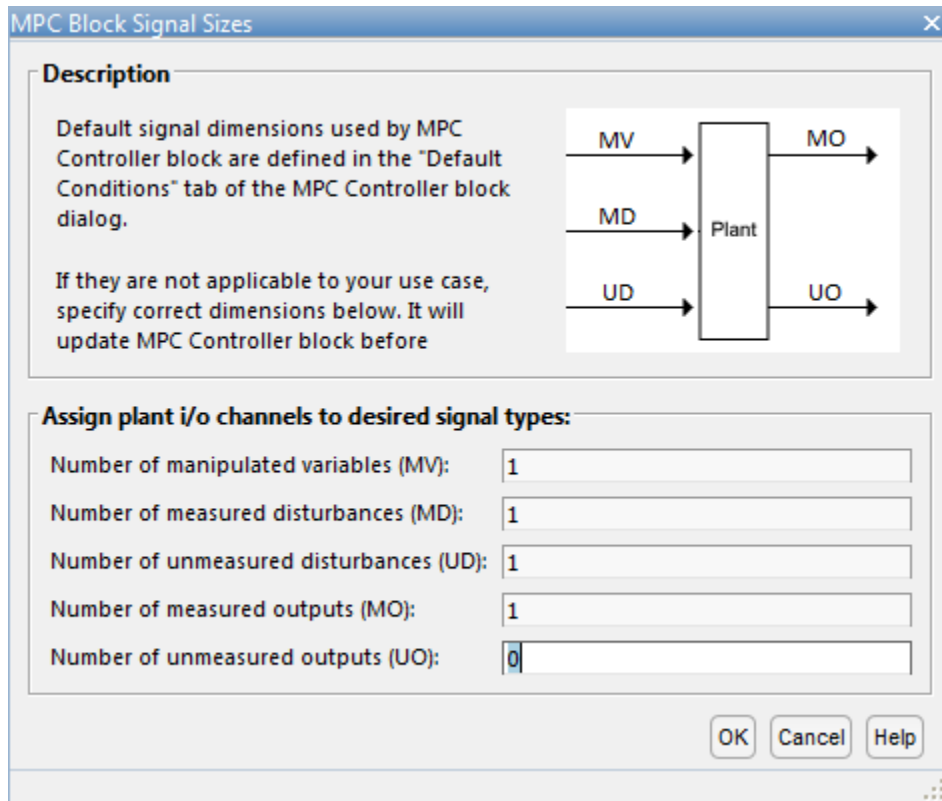
Click **OK**.

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)**.

Define MPC Structure By Linearization

MPC Structure

	1 Measured Disturbances		**0** Unmeasured	
Setpoints (reference) → MPC	**1** Manipulated Variables → Inputs → Plant		Outputs	Change I/O Sizes
	1 Unmeasured Disturbances		**1** Measured	

Controller Sample Time

Specify MPC controller sample time (default sample time in the MPC block): `0.1`

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
○	Measured Disturbances (MD)	CSTR_ClosedLoop/Feed Temperature:1	
○	Unmeasured Disturbances (UD)	Select a UD signal in the Simulink model	

Simulink Signals for Plant Outputs

| Selected	Type	Block Path	
⦿	Measured Outputs (MO)	CSTR_ClosedLoop/CSTR:2	Select Signals

Define and Linearize   Cancel   Help

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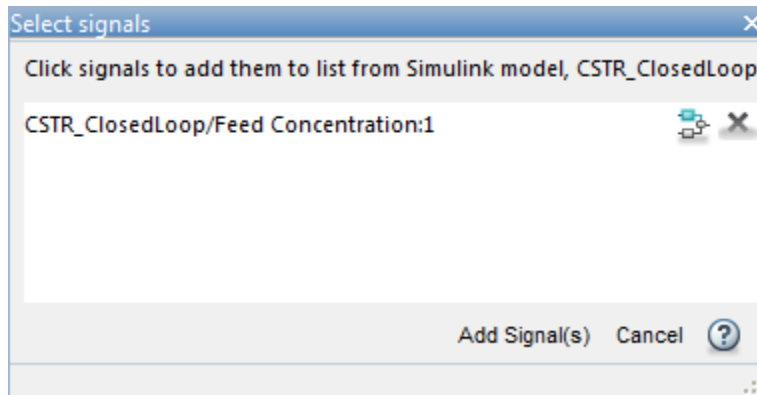
In the Select Signals dialog box, click **Add Signal(s)**.

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 click **Linear Analysis Points** > **Trim Output Constraint**.
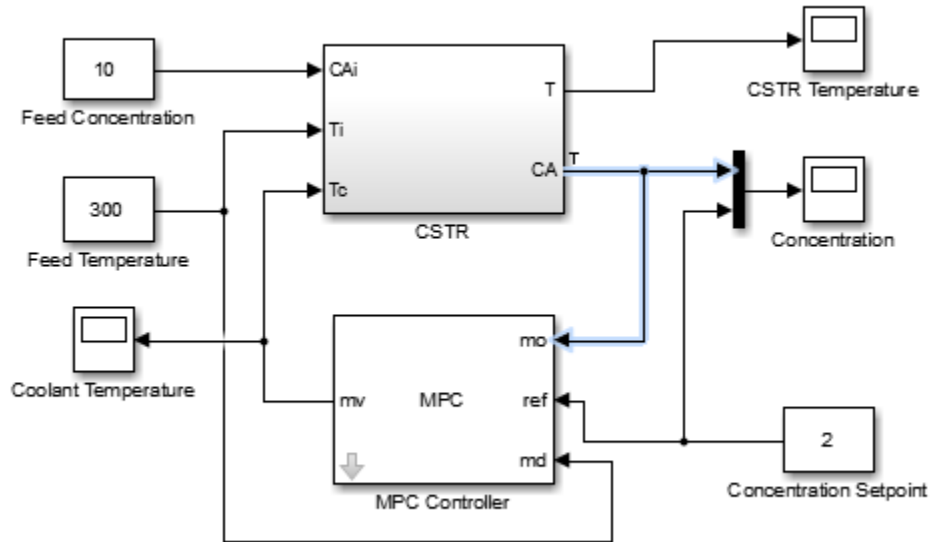
The CA signal can now be used to define output specifications for calculating a model steady-state operating point.

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.
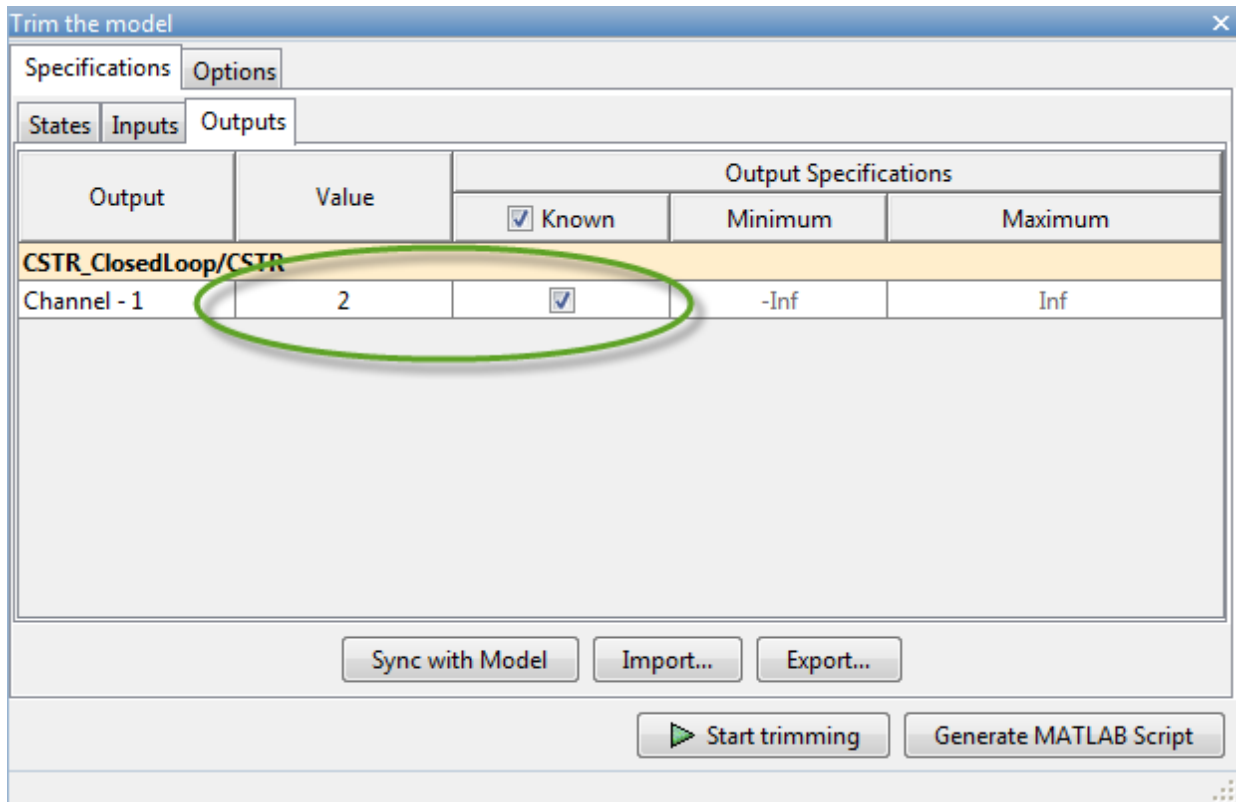
This setting constrains the value of the output signal during the operating point search to a known value.

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**.

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 MPC Designer **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
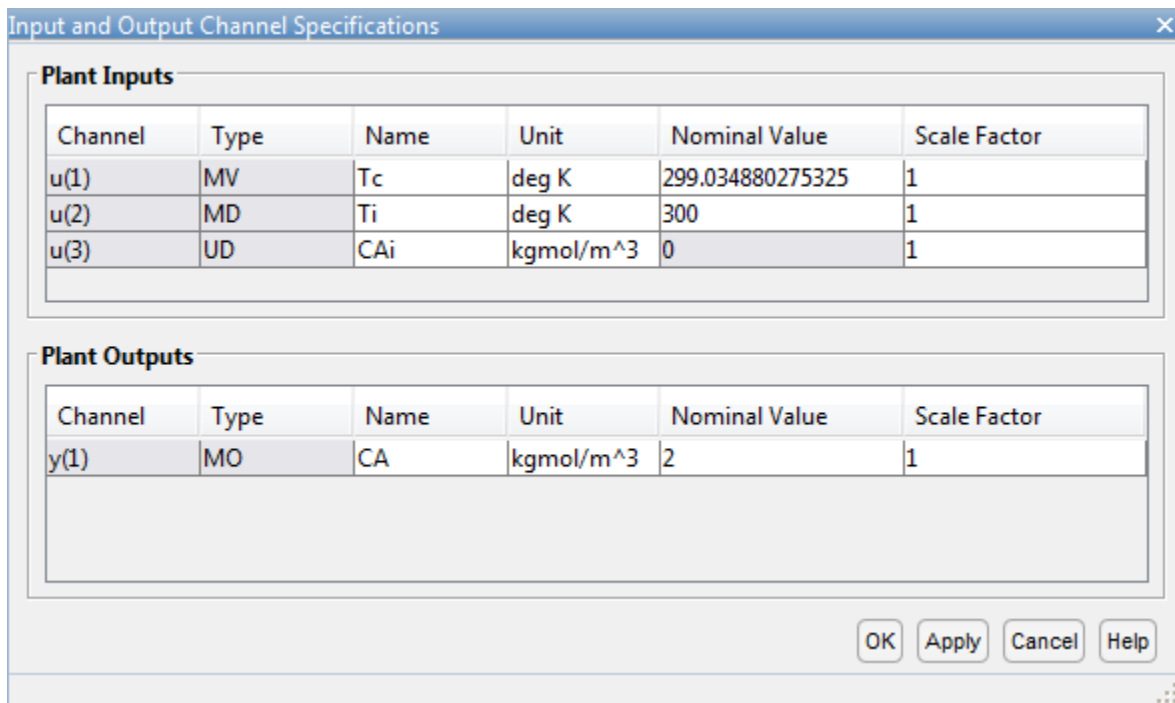
- A default simulation scenario

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**.

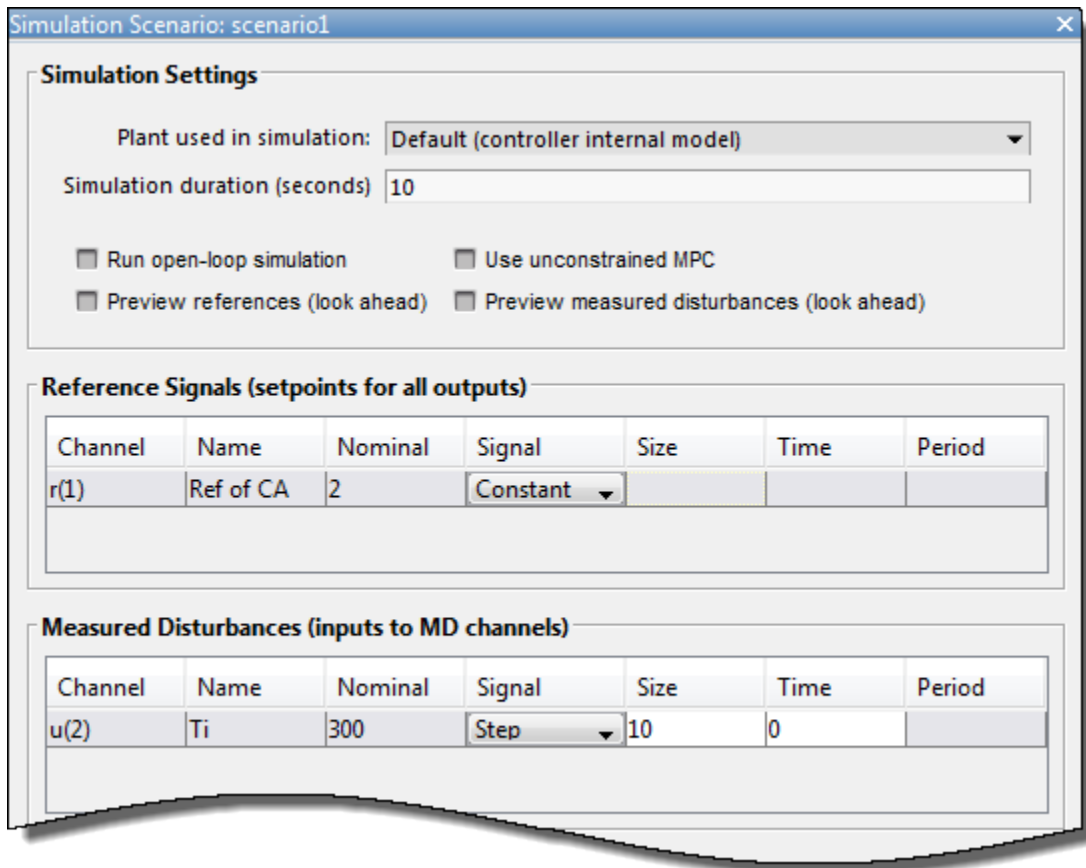**Define Disturbance Rejection Simulation Scenarios**

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.

Simulation Scenario: scenario1

Simulation Settings

Plant used in simulation: Default (controller internal model)

Simulation duration (seconds) 10

☐ Run open-loop simulation   ☐ Use unconstrained MPC
☐ Preview references (look ahead)   ☐ Preview measured disturbances (look ahead)

Reference Signals (setpoints for all outputs)

Channel	Name	Nominal	Signal	Size	Time	Period
r(1)	Ref of CA	2	Constant ▾			

Measured Disturbances (inputs to MD channels)

| Channel	Name	Nominal	Signal	Size	Time	Period
u(2)	Ti	300	Step ▾	10	0	

Click **OK**.

In the **Data Browser**, under **Scenarios**, double-click `scenario1` 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.
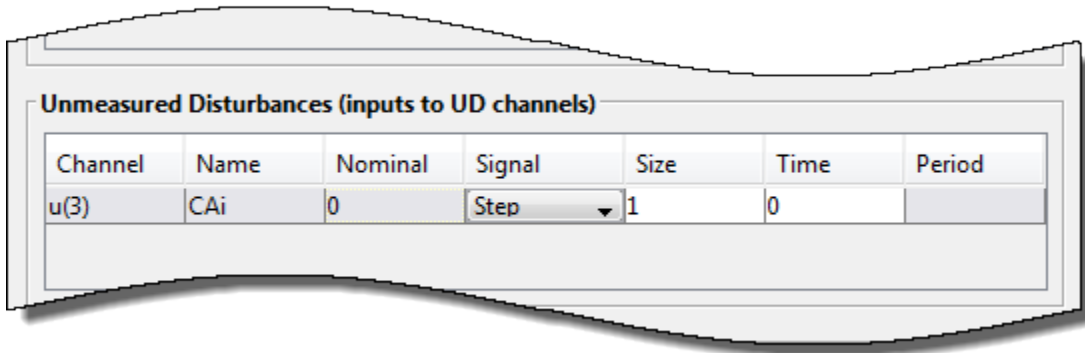
Click **OK**.

In the **Data Browser**, under **Scenarios**, double-click `NewScenario` and rename it `UD_reject`.
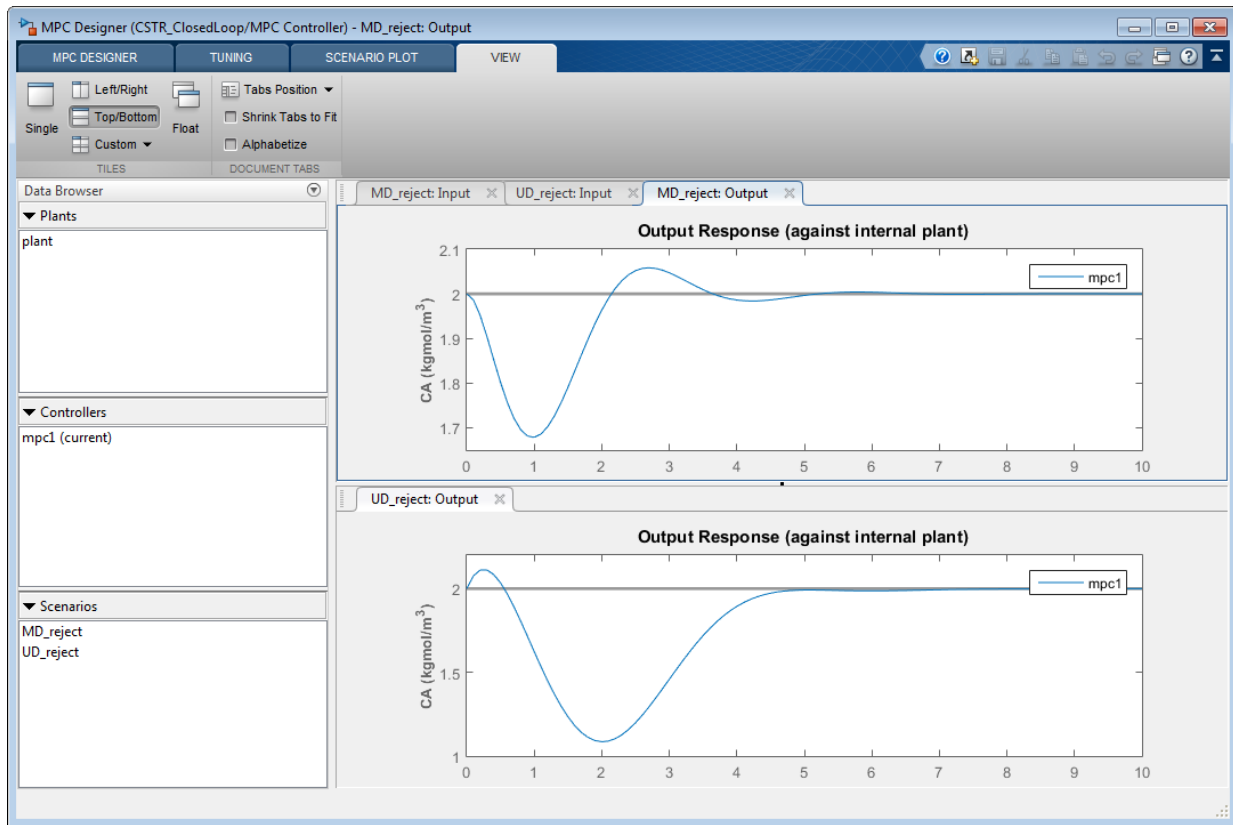
Arrange Output Response Plots

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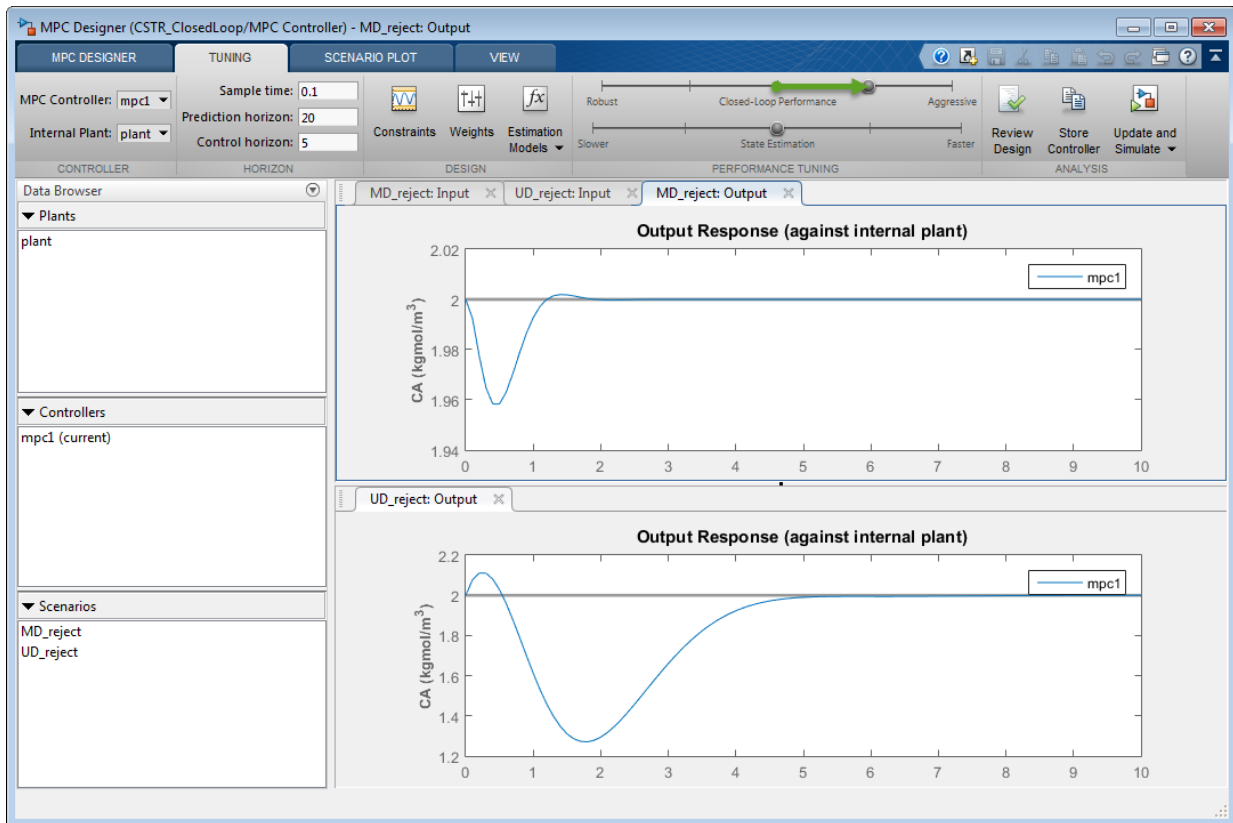
Drag the **State Estimation** slider to the right, which leads to more aggressive unmeasured disturbance rejection. Drag the slider until the **UD_reject: Output** response reaches steady state in less than 3 seconds.
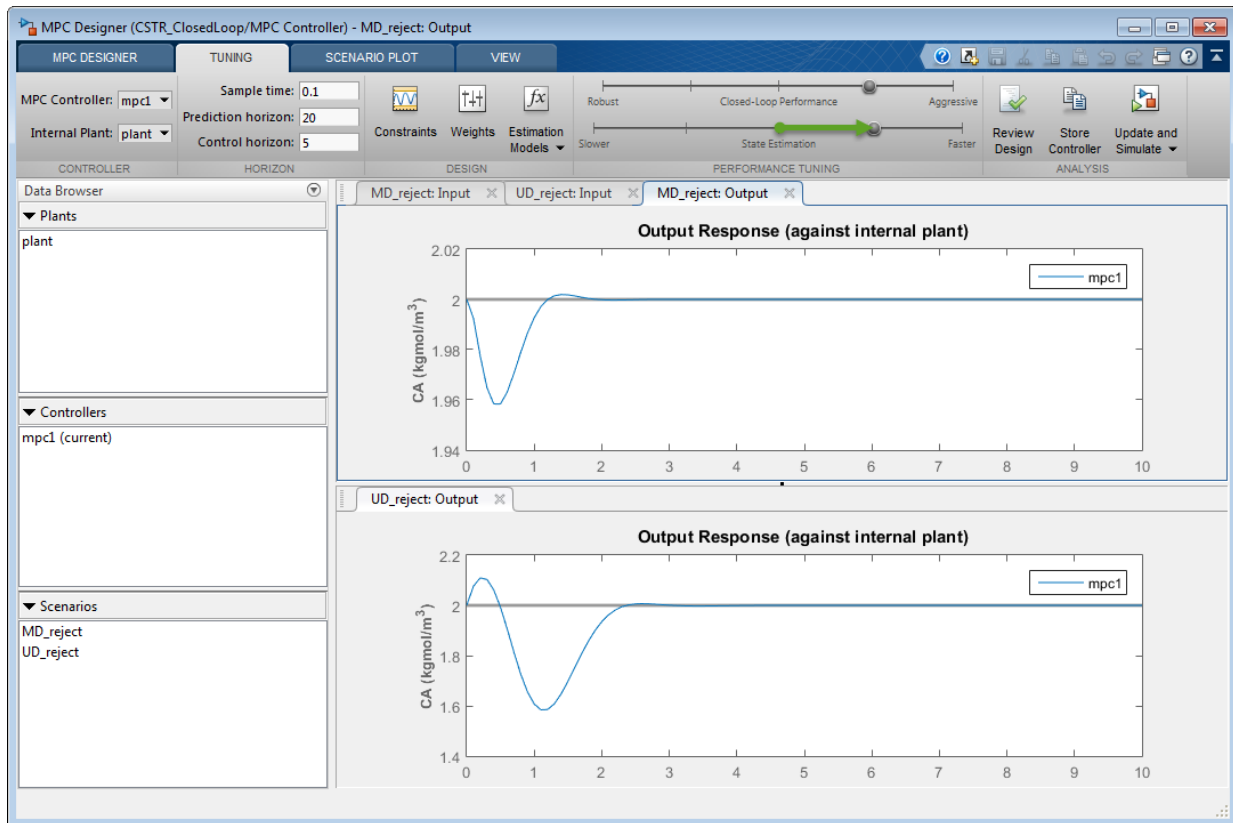
Update Simulink Model with Tuned Controller

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.

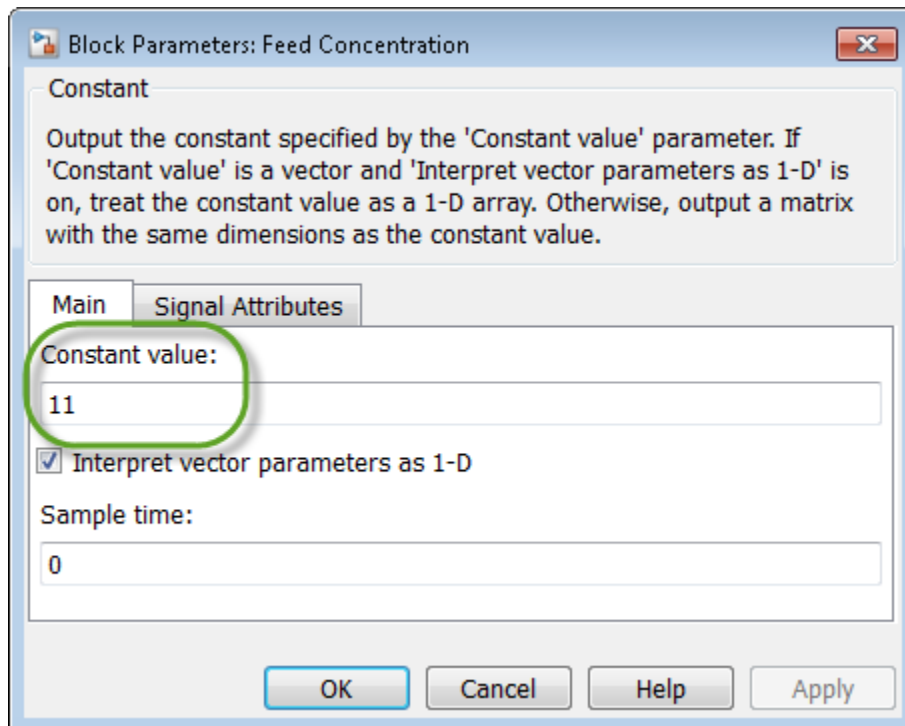### Simulate Unmeasured Disturbance Rejection

In the Simulink model window, change the simulation duration to 5 seconds.
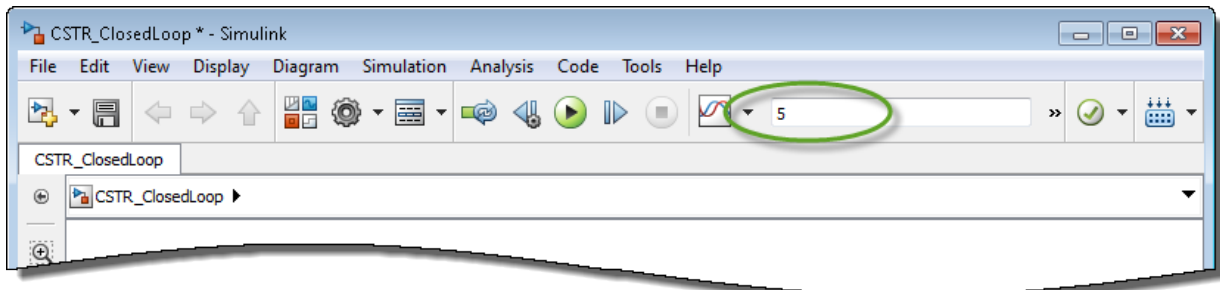
The model initial conditions are set to the nominal operating point used for linearization.
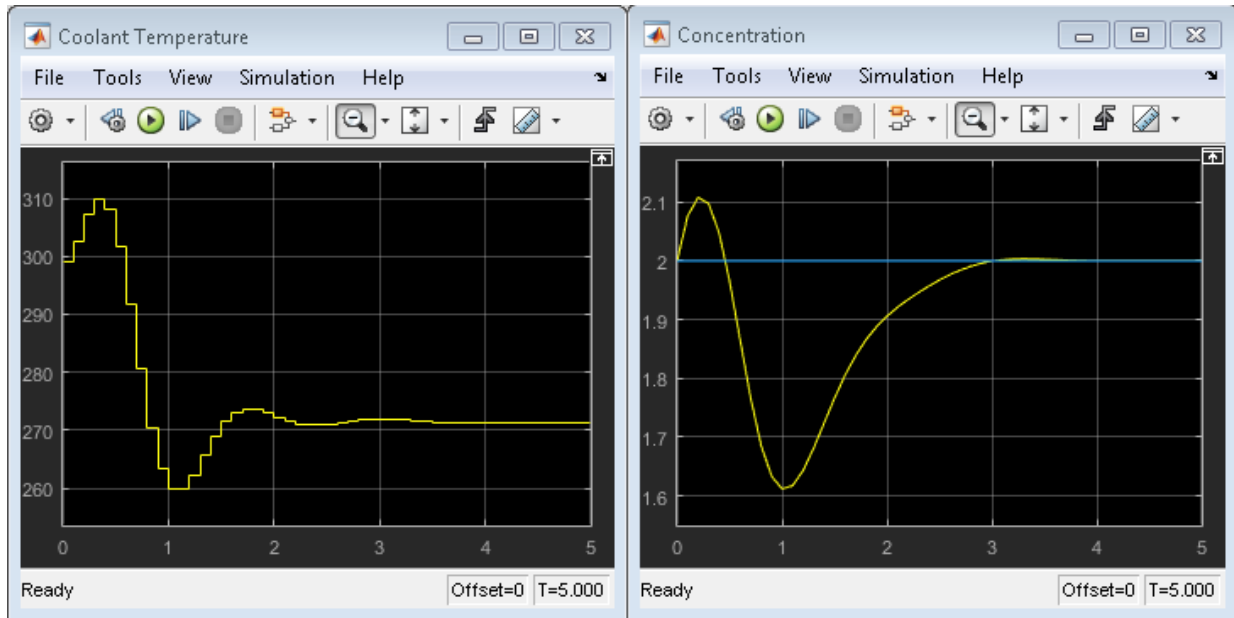
Double-click the **Feed Concentration** block.

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.
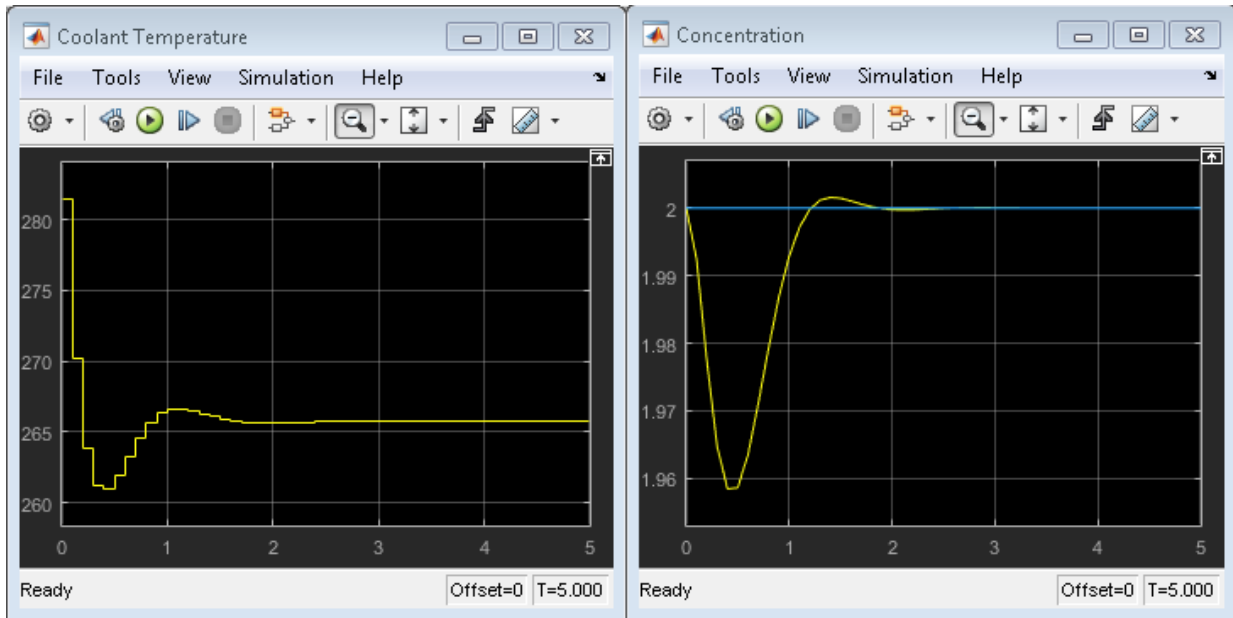
Click **OK**.

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

Related Examples

- "Linearize Simulink Models" on page 2-22
- "Design Controller Using MPC Designer" on page 3-2

More About

- "Tuning Weights"

Test an Existing Controller

This topic shows how to test an existing model predictive controller by adding it to a Simulink model.

1   Open your Simulink model.

2   Add an `MPC Controller` block to the model.

3   If your controller includes measured disturbances, add the `md` inport to the `MPC Controller` block.
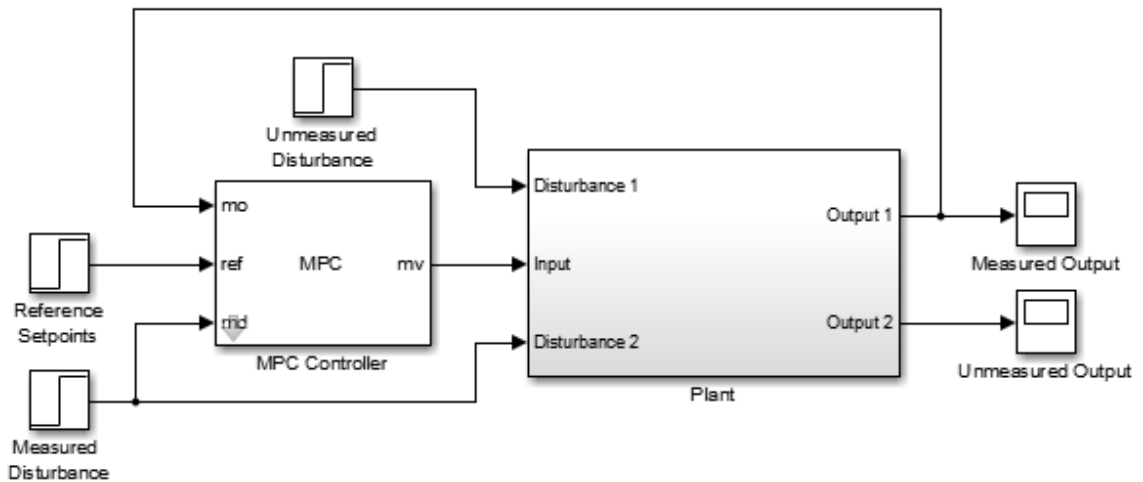
Double-click the `MPC Controller` block.

In the Block Parameters dialog box, on the **General** tab, select **Measured disturbance (md)**.
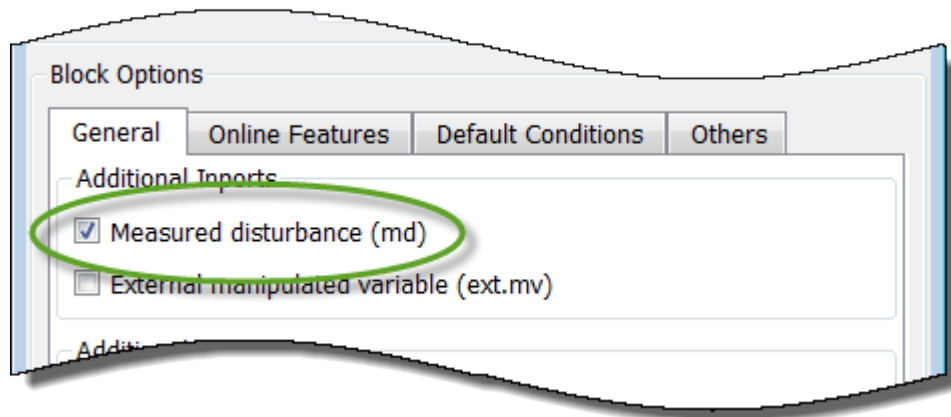
Click **OK**.

4   Connect the plant and controller signals in the Simulink model. Connect:

- The plant inputs to the manipulated variable (`mv`) inport of the `MPC Controller` block.

- The plant measured outputs to the measured output (`mo`) inport of the `MPC Controller` block.

- The measured disturbances, if any, to the plant and to the measured disturbance (`md`) inport of the `MPC Controller` block.

- Any unmeasured disturbances or unmeasured outputs to their corresponding plant inport and outport.
- The reference signals to the reference (ref) inport of the MPC Controller block.

5 Specify the controller.

Double-click the MPC Controller block.

In the Block Parameters dialog box, in the **MPC Controller** field, specify the name of an mpc controller from the MATLAB workspace.
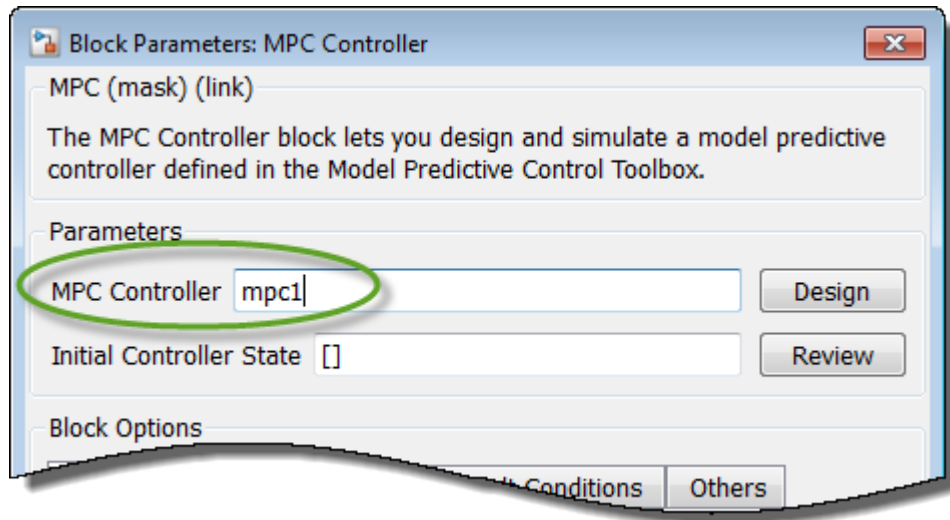
Click **OK**.

6 (Optional) Modify the controller.

After specifying a controller in the `MPC Controller` block, you can modify the controller:

- Using the MPC Designer app:

  - In the Block Parameters dialog box, click **Design**.
  - In the MPC Designer app, tune the controller parameters.
  - In the **MPC Designer** tab, in the **Result** section, click **Update and Simulate** > **Update Block Only**.

    The app exports the updated controller to the MATLAB workspace.

- Using commands to modify the controller object in the MATLAB workspace.

7 Run the Simulink model.

Tip If you do not have a Simulink model of your plant, you can generate one that uses your MPC controller to control its internal plant model. For more information, see "Generate Simulink Model from MPC Designer".

See Also
mpc | MPC Controller | MPC Designer

Related Examples

- "Design MPC Controller in Simulink" on page 5-2
- "Design MPC Controller at the Command Line" on page 4-2
- "Generate Simulink Model from MPC Designer"

Schedule Controllers at Multiple Operating Points

| In this section... |
|---|
| "A Two-Model Plant" on page 5-27 |
| "Animation of the Multi-Model Example" on page 5-28 |
| "Designing the Two Controllers" on page 5-29 |
| "Simulating Controller Performance" on page 5-30 |

If your plant is nonlinear, a controller designed to operate in a particular target region may perform poorly in other regions. The common way to compensate is to design multiple controllers, each intended for a particular combination of operating conditions. Switch between them in real-time as conditions change. Gain scheduling is a conventional example of this technique. The following example shows how to coordinate multiple model predictive controllers for this purpose.

The rest of this section is an illustrative example organized as follows:

- "A Two-Model Plant" on page 5-27 describes the example application involving two distinct operating conditions.
- "Animation of the Multi-Model Example" on page 5-28 explains how to simulate the plant.
- "Designing the Two Controllers" on page 5-29 shows how to obtain the controllers for each operating condition.
- "Simulating Controller Performance" on page 5-30 shows how to use the `Multiple MPC Controllers` block in Simulink to coordinate the two controllers.

A Two-Model Plant

The figure below is a stop-action snapshot of the plant. It consists of two masses, `M1` (upper) and `M2` (lower). A spring connects `M1` to a rigid wall and pulls it to the right. An applied force, shown as a red arrow, opposes the spring pulling `M1` to the left.
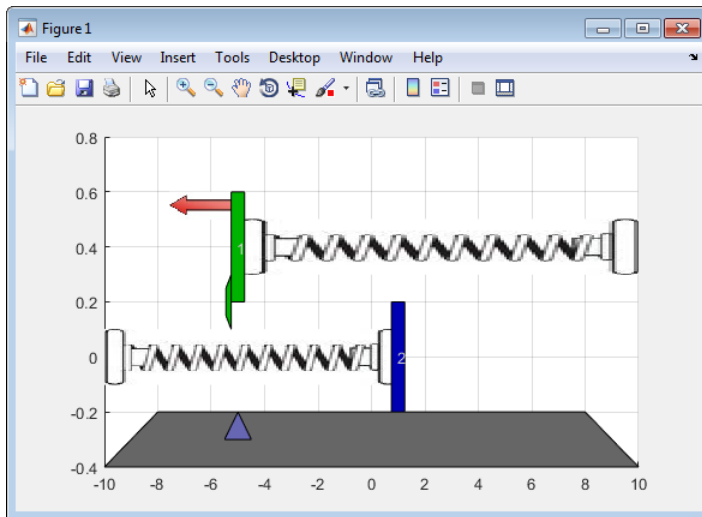
In the above, mass M2 is uncontrollable. It responds solely to the spring pulling it to the left.

If the two masses collide, however, they stick together until a change in the applied force separates them.

The control objective is to make M1 track a reference signal. The blue triangle in the above indicates the desired location, which varies with time. At the instant shown, the desired numeric location is -5.

For an animated version of the plant, see Scheduling Controllers for a Plant with Multiple Operating Points.

Animation of the Multi-Model Example

In order to achieve its objective, the controller can adjust the applied force magnitude (the length of the red arrow). It receives continuous measurements of the M1 location. There is also a contact sensor to signal collisions. The M2 location is unmeasured.

If M1 were isolated, this would be a routine control problem. The challenge is that the relationship between the applied force and M1's movement changes dramatically when M2 attaches to M1.

Define the model.

Define the system parameters.

```
M1 = 1;      % mass
M2 = 5;      % mass
k1 = 1;      % spring constant
k2 = 0.1;    % spring constant
b1 = 0.3;    % friction coefficient
b2 = 0.8;    % friction coefficient
yeq1 = 10;   % wall mount position
yeq2 = -10;  % wall mount position
```

Define a model of M1 when the masses are separated. Its states are the position and velocity of M1. Its inputs are the applied force, which will be the controller's manipulated variable (MV), and a spring constant calibration signal, which a measured disturbance (MD) input.

```
A1 = [0 1;-k1/M1 -b1/M1];
B1 = [0 0;-1/M1 k1*yeq1/M1];
C1 = [1 0];
D1 = [0 0];
sys1 = ss(A1,B1,C1,D1);
sys1 = setmpcsignals(sys1,'MV',1,'MD',2);
```

The call to setmpcsignals specifies the input type for the two inputs.

Define another model (with the same input/output structure) to predict movement when the two masses are joined.

```
A2 = [0 1;-(k1+k2)/(M1+M2) -(b1+b2)/(M1+M2)];
B2 = [0 0;-1/(M1+M2) (k1*yeq1+k2*yeq2)/(M1+M2)];
C2 = [1 0];
D2 = [0 0];
sys2 = ss(A2,B2,C2,D2);
sys2 = setmpcsignals(sys2,'MV',1,'MD',2);
```

Designing the Two Controllers

Next, define controllers for each case. Both controllers employ a 0.2 second sampling period (Ts), a prediction horizon of $p = 20$, a control horizon of $m = 1$ and default values for all other controller parameters.

The only property that distinguishes the two controllers is the prediction model.

```
Ts = 0.2;
p = 20;
m = 1;
MPC1 = mpc(sys1,Ts,p,m); % Controller for M1 detached from M2
MPC2 = mpc(sys2,Ts,p,m); % Controller for M1 connected to M2
```

The applied force also have the same constraints in each case. The lower bound for the applied force is zero because it cannot reverse direction. The maximum rate of change for the applied force is 1000 per second (increasing or decreasing).

```
MPC1.MV = struct('Min',0,'RateMin',-1e3,'RateMax',1e3);
MPC2.MV = struct('Min',0,'RateMin',-1e3,'RateMax',1e3);
```

Simulating Controller Performance

The following is the Simulink block diagram for this example.
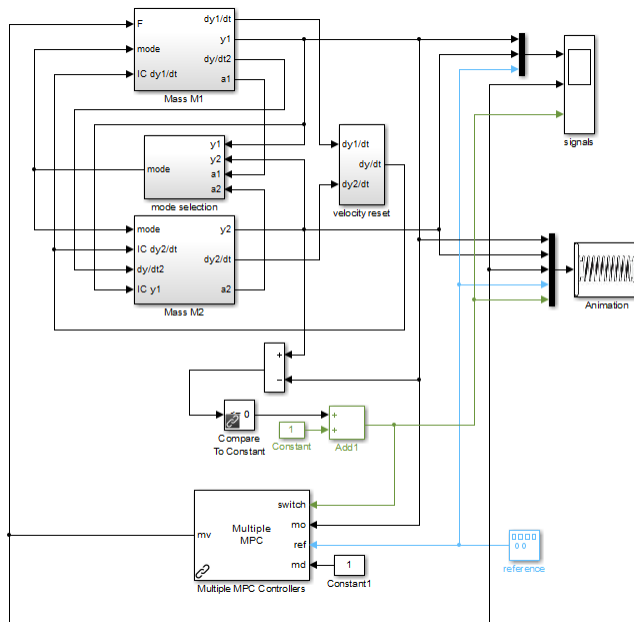

The upper portion simulates the movement of the two masses, plots the signals as a function of time and animates the plant.

The lower part contains the following key elements:

- A pulse generator that supplies the desired M1 position (the controller reference signal). The output of the pulse generator is a square wave that varies between —5 and 5, with a frequency of 0.015 per second.

- Simulation of a contact sensor. When the two masses have the same position, the `Compare to Constant` block evaluates to `true` and the Add1 block converts this to a 2. Else, the output of Add1 is 1.

- The `Multiple MPC Controllers` block, which has four inputs. The measured output (`mo`), reference (`ref`) and measured disturbance (`md`) inputs are as for the `MPC Controller` block. The distinctive feature of the `Multiple MPC Controllers` block is the `switch` input.

 The figure below shows the `Multiple MPC Controllers` block mask for this example.
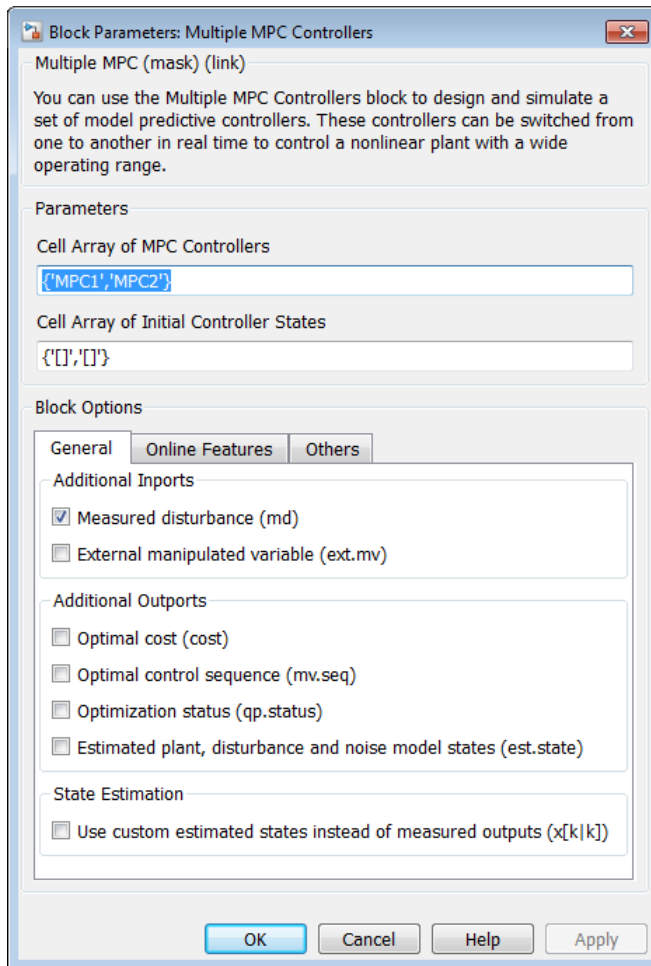
When the `switch` input is 1, the block automatically activates the first controller that is listed (`MPC1`), which is appropriate when the masses are separated. When the `switch` input is 2, the block automatically enables the second controller (`MPC2`).

Simulate the controller performance using Simulink commands.

```
Tstop = 100;    % Simulation time
y1initial = 0; % Initial M1 and M2 Positions
```

```
y2initial = 10;
open('mpc_switching')
sim('mpc_switching',Tstop)
```

The figure below shows the `signals` scope output.
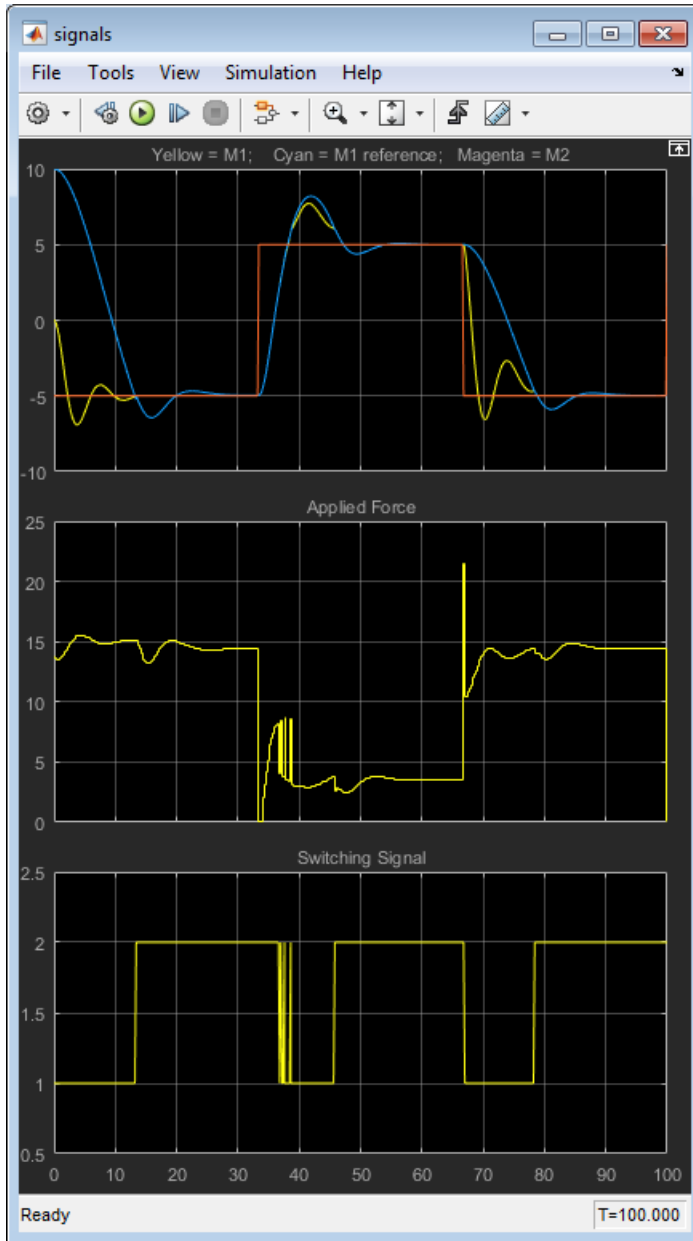
In the upper plot, the cyan curve is the desired position. It starts at -5. The M1 position (yellow) starts at 0. Under the control of MPC1, M1 moves rapidly towards the desired position. M2 (magenta) starts at 10 and begins moving in the same direction.

At about t = 13 seconds, M2 collides with M1. The switching signal (lower plot) changes at this instant from 1 to 2, so MPC2 takes over.

The completely inelastic collision moves M1 away from its desired position and M2 remains joined to M1. Controller MPC2 adjusts the applied force (middle plot) so M1 quickly returns to the desired position.

When the desired position changes step-wise to 5, the two masses separate briefly, with the controller switching to MPC1 appropriately. But, for the most part, the two masses move together and settle rapidly at the desired location. The transition back to —5 is equally well behaved.

Suppose we force MPC2 to operate under all conditions.

The figure below shows the result.
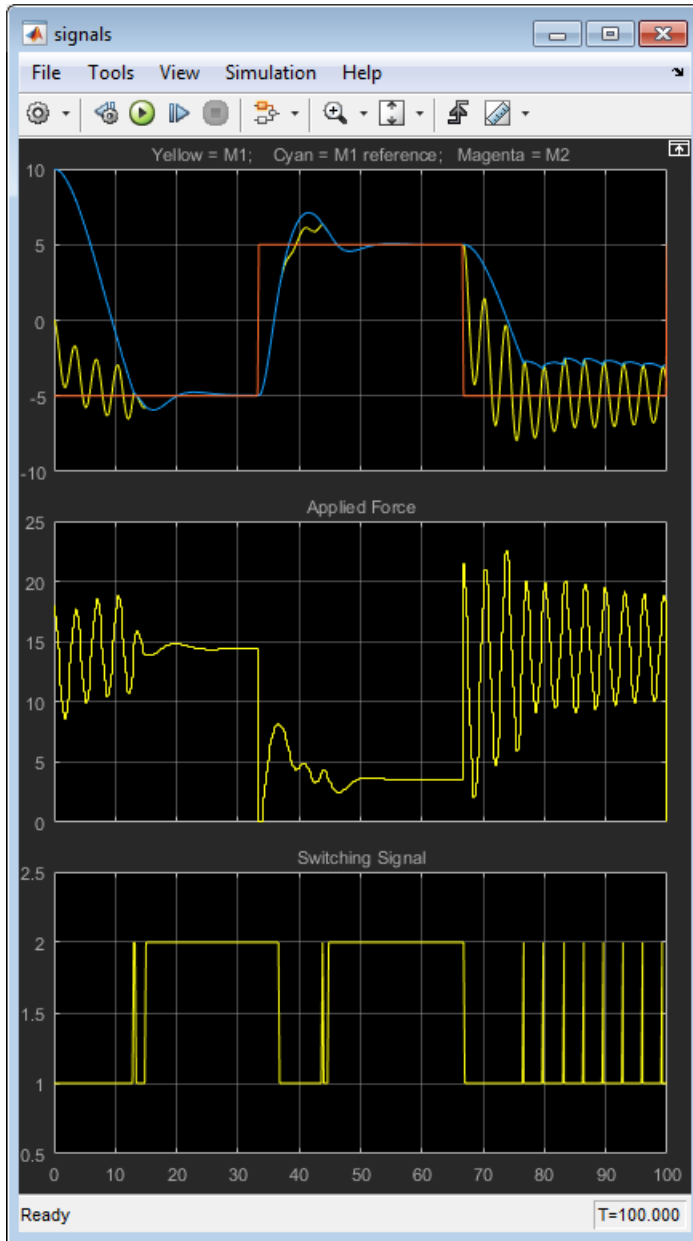
When the masses are disconnected, as at the start, MPC2 applies excessive force and then overcompensates, resulting in oscillatory behavior. Once the masses join, the move more smoothly, as would be expected.

The last transition causes especially severe oscillations. The masses collide frequently and M1 never reaches the desired position.

If MPC1 is in charge exclusively, the masses move sluggishly and fail to settle at the desired position before the next transition occurs. For more information, see the Scheduling Controllers for a Plant with Multiple Operating Points.

In this application, at least, two controllers are, indeed, better than one.