Model Predictive Control Toolbox™
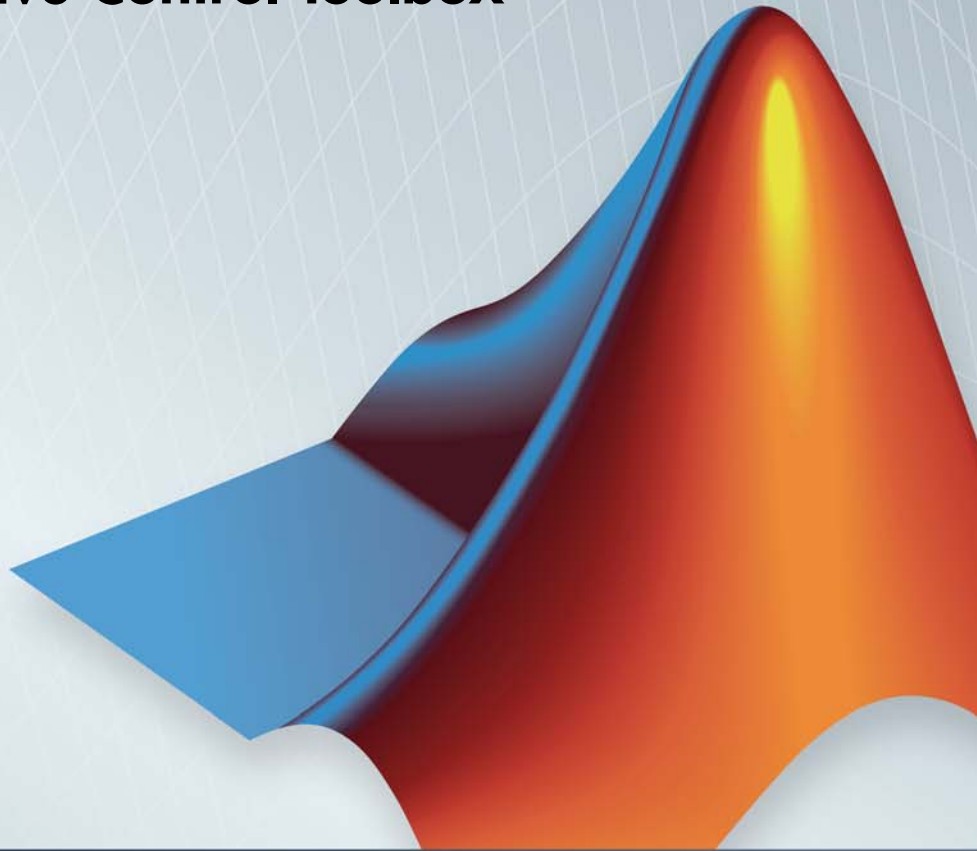
User's Guide

R2013a

Alberto Bemporad
Manfred Morari
N. Lawrence Ricker

MATLAB®

MathWorks®

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Model Predictive Control Toolbox™ User's Guide

© COPYRIGHT 2005–2013 by The MathWorks, Inc.

Trademarks

Patents

Revision History

October 2004	First printing	New for Version 2.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 2.2.2 (Release 2006a)
September 2006	Online only	Revised for Version 2.2.3 (Release 2006b)
March 2007	Online only	Revised for Version 2.2.4 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 2.3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.1.1 (Release 2009b)
March 2010	Online only	Revised for Version 3.2 (Release 2010a)
September 2010	Online only	Revised for Version 3.2.1 (Release 2010b)
April 2011	Online only	Revised for Version 3.3 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)
March 2012	Online only	Revised for Version 4.1 (Release 2012a)
September 2012	Online only	Revised for Version 4.1.1 (Release 2012b)
March 2013	Online only	Revised for Version 4.1.2 (Release R2013a)

Contents

Model Predictive Control Simulink Library

3

Case-Study Examples

4

Reference for the Design Tool GUI

5

Index

1

Introduction

- "How Plants Are Used In Model Predictive Control" on page 1-2
- "Blocking" on page 1-12
- "Typical Sampling Instant" on page 1-14
- "Prediction and Control Horizons" on page 1-18

How Plants Are Used In Model Predictive Control

In this section...

Single-Input Single-Output (SISO) Plants

The typical Model Predictive Control Toolbox™ application involves a plant having multiple inputs and multiple outputs (*MIMO*).

Consider the simpler single-input single-output (SISO) application shown in the following figure.
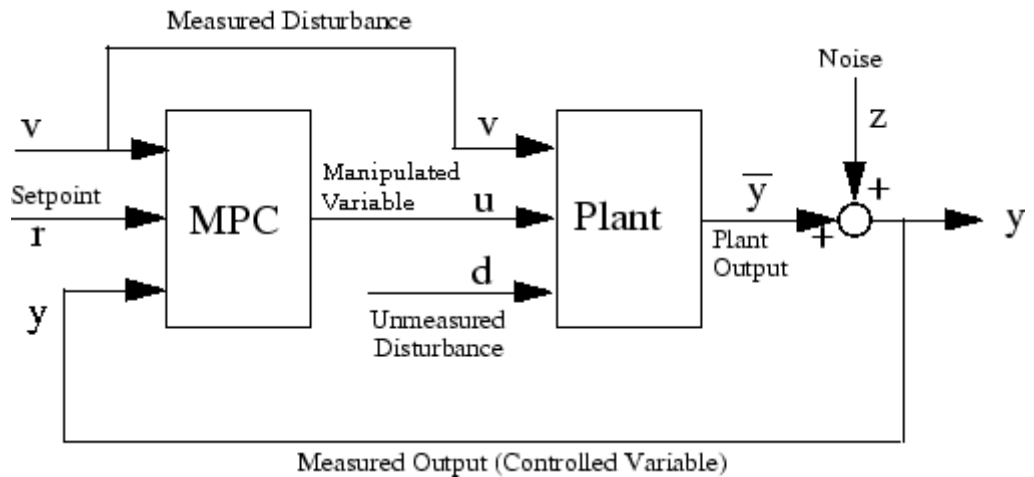
Block Diagram of a SISO Model Predictive Control Toolbox™ Application

The plant can be a manufacturing process, such as a unit operation in an oil refinery, or a device, such as an electric motor. The main objective is to hold a single *output*, $\bar{y}$, at a *reference value* (or *setpoint*), *r*, by adjusting a single *manipulated variable* (or *actuator*) *u*. This is what is generally termed a *SISO* (single-input single-output) plant. The block labeled MPC represents a Model Predictive Controller designed to achieve the control objective.

The following table summarizes the nomenclature used in the previous figure.

Model Predictive Control Toolbox Signals

Symbol	Description
d	*Unmeasured disturbance.* This signal is unknown except for its effect on the plant output. The controller provides *feedback* compensation for such disturbances.
r	*Setpoint* (or *reference*). The target value for the output.
u	*Manipulated variable (or actuator*). The signal the controller adjusts in order to achieve its objectives.
v	*Measured disturbance* (*optional*). The controller provides *feedforward* compensation for such disturbances as they occur to minimize their impact on the output.
$\bar{y}$	*Output* (or *controlled variable*). The signal to be held at the setpoint. This is the true value, uncorrupted by measurement noise.
y	*Measured output.* The signal used to estimate the true value, $\bar{y}$.
z	*Measurement noise.* The signal represents electrical noise, sampling errors, drifting calibration, and other effects that impair measurement precision and accuracy.

The SISO plant actually has multiple inputs. In addition to the manipulated variable input, u, there may be:

- An *unmeasured disturbance, d*. The unmeasured disturbance is always present. It is an *independent* input—not affected by the controller or the plant. It represents all the unknown, unpredictable events that upset plant operation. In the context of Model Predictive Control, it can also represent unmodeled dynamics. When such an event occurs, the only indication is its effect on the *measured* output, y, which is *fed back* to the controller, as shown in the previous figure. Some applications have unmeasured disturbances only.

- A *measured disturbance, v*. It is an independent input that affects $\bar{y}$. In contrast to *d*, the controller receives the measured *v* directly, as shown in the previous figure. Receiving the signal directly allows the controller to compensate for *v*'s impact on $\bar{y}$ immediately rather than waiting until the effect appears in the *y* measurement. This approach is called *feedforward* control.

Model Predictive Control Toolbox design always provides *feedback* compensation for unmeasured disturbances and *feedforward* compensation for any measured disturbance.

The design requires a *model* of the impact that *v* and *u* have on $\bar{y}$ (symbolically, $v \rightarrow \bar{y}$ and $u \rightarrow \bar{y}$). It uses this *plant model* to calculate the *u* adjustments needed to keep $\bar{y}$ at its setpoint.

This calculation considers the effect of any known constraints on the adjustments (typically an actuator upper or lower bound or a constraint on how rapidly *u* can vary). You can also specify bounds on $\bar{y}$. The ability to create such constraint specifications is a distinguishing feature of Model Predictive Control Toolbox design and can be particularly valuable when you have multiple control objectives to be achieved *via* multiple adjustments (a MIMO plant). In the context of a SISO system, such constraint handling is often termed as *anti-windup*.

If the plant model is accurate, the plant responds quickly to adjustments in *u*, and no constraints are encountered, feedforward compensation can counteract the impact of *v* perfectly. In reality, model imperfections, physical limitations, and unmeasured disturbances cause the *y* to deviate from its setpoint. Therefore, Model Predictive Control Toolbox design includes a *disturbance model* ($d \rightarrow \bar{y}$) to *estimate d* and *predict* its impact on $\bar{y}$. It then uses its $u \rightarrow \bar{y}$ model to calculate appropriate adjustments (feedback). This calculation also considers the known constraints.

Various *noise* effects can corrupt the measurement. The signal *z* in the Block Diagram of a SISO Model Predictive Control Toolbox™ Application on page 1-2 figure represents such effects. Theses effects can vary randomly with a zero mean, or can exhibit a nonzero, drifting bias. Model Predictive Control

Toolbox design uses a $z \rightarrow y$ model in combination with its $d \rightarrow \bar{y}$ model to remove the estimated noise component (*filtering*).

The preceding feedforward/feedback actions comprise the controller's *regulator* mode. The design also provides a *servo* mode, *i.e.*, it adjusts u such that $\bar{y}$ tracks a time-varying setpoint.

The tracking accuracy depends on the plant characteristics (including constraints), the accuracy of the $u \rightarrow \bar{y}$ model, and whether or not future setpoint variations can be *anticipated*, *i.e.*, known in advance. If so, it provides feedforward compensation for these.

Multi-Input Multi-Output (MIMO) Plants

One advantage of Model Predictive Control Toolbox design (relative to classical multi-loop control) is that it generalizes directly to plants having multiple inputs and outputs. Moreover, the plant can be *non-square*, i.e., having an unequal number of actuators and outputs. Industrial applications involving hundreds of actuators and controller outputs have been reported.

The main challenge is to tune the controller to achieve multiple objectives. For example, if there are several outputs to be controlled, it might be necessary to prioritize so that the controller provides accurate setpoint tracking for the most important output, sacrificing others when necessary, e.g., when it encounters constraints. Model Predictive Control Toolbox features support such prioritization.

Optimization and Constraints

As discussed in more detail in "Optimization Problem" on page 2-5, the Model Predictive Control Toolbox controller solves an optimization problem much like the LQG optimal control described in the Control System Toolbox™ product. The main difference is that the Model Predictive Control Toolbox optimization problem includes explicit *constraints* on u and y, and it optimizes over a finite horizon.

Setpoint Tracking. Consider first a case with no constraints. A primary control objective is to force the plant outputs to track their setpoints.

Specifically, the controller predicts how much each output will deviate from its setpoint within the prediction horizon. It multiplies each deviation by the output's weight, and computes the weighted sum of squared deviations, $S_y(k)$, as follows:

$$S_y(k) = \sum_{i=1}^{P} \sum_{j=1}^{n_y} \left\{ w_j^y [r_j(k+i) - y_j(k+i)] \right\}^2$$

where k is the current sampling interval, $k+i$ is a future sampling interval (within the prediction horizon), P is the prediction horizon, n_y is the number

of plant outputs, w_j^y is the *weight* for output j, and $[r_j(k+i) - y_j(k+i)]$ is the predicted deviation at future instant $k+i$.

If $w_j^y \ \square \ w_{i \neq j}^y$, the controller does its best to track r_j, sacrificing r_i tracking

if necessary. If $w_j^y = 0$, on the other hand, the controller completely ignores deviations $r_j - y_j$.

Choosing the weights is a critical step. You will usually need to tune your controller, varying the weights to achieve the desired behavior.

As an example, consider the following figure, which depicts a type of chemical reactor (a CSTR). Feed enters continuously with reactant concentration C_{Ai}. A reaction takes place inside the vessel at temperature T. Product exits continuously, and contains residual reactant at concentration C_A ($<C_{Ai}$).

The reaction liberates heat. A coolant having temperature T_c flows through coils immersed in the reactor to remove excess heat.
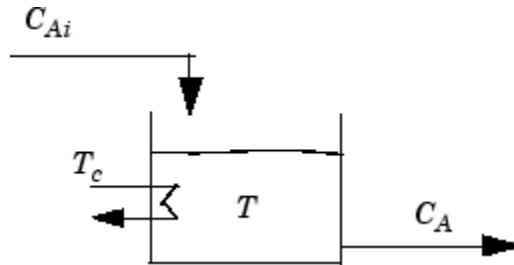
CSTR Schematic

From the Model Predictive Control Toolbox point of view, T and C_A would be plant outputs, and C_{Ai} and T_c would be inputs. More specifically, C_{Ai} would be an independent disturbance input, and T_c would be a manipulated variable (actuator).

There is one manipulated variable (the coolant temperature), so it's impossible to hold both T and C_A at setpoints. Controlling T would usually be a high priority. Thus, you might set the output weight for T much larger than that for C_A. In fact, you might set the C_A weight to zero, allowing C_A to move freely within an *acceptable operating region* (to be defined by constraints).

Move Suppression. If the controller focuses exclusively on setpoint tracking, it might choose to make large manipulated-variable adjustments. These could be impossible to achieve. They could also accelerate equipment wear or lead to control system instability.

Thus, the Model Predictive Controller also monitors a weighted sum of controller adjustments, calculated according to the following equation:

$$S_{\Delta u}(k) = \sum_{i=1}^{M} \sum_{j=1}^{n_{mv}} \left\{ w_j^{\Delta u} \Delta u_j(k+i-1) \right\}^2$$

where M is the control horizon, n_{mv} is the number of manipulated variables, $\Delta u_j(k+i-1)$ is the predicted adjustment (i.e., *move*) in manipulated variable

j at future (or current) sampling interval $k+i-1$, and $w_j^{\Delta u}$ is a weight,

which must be zero or positive. Increasing $w_j^{\Delta u}$ forces the controller to make

smaller, more cautious Δu_j moves. In many cases (but not all) this will have the following effects:

- The controller's setpoint tracking will degrade.

- The controller will be less sensitive to prediction inaccuracies (i.e., more *robust*).

Setpoints on Manipulated Variables. In most applications, the controller's manipulated variables (MVs) should move freely (within a constrained region) to compensate for disturbances and stepoint changes. An attempt to hold an MV at a point within the region would degrade output setpoint tracking.

On the other hand, some plants have more MVs than output setpoints. In such a plant, if all manipulated variables were allowed to move freely, the MV values needed to achieve a particular setpoint or to reject a particular disturbance would be non-unique. Thus, the MVs would drift within the operating space.

A common approach is to define setpoints for "extra" MVs. These setpoints usually represent operating conditions that improve safety, economic return, etc. Model Predictive Control Toolbox design includes an additional term to accommodate such cases, as follows:

$$S_u(k) = \sum_{i=1}^{M} \sum_{j=1}^{n_{mv}} \left\{ w_j^u [\bar{u}_j - u_j(k+i-1)] \right\}^2$$

where $\bar{u}_j$ is the manipulated variable setpoint (nominal value) for the jth MV,

and w_j^u is the corresponding weight.

Constraints. Constraints may be either *hard* or *soft*. A hard constraint must not be violated. Unfortunately, under some conditions a constraint violation might be unavoidable (e.g., an unexpected, large disturbance), and a realistic controller must allow for this.

Model Predictive Control Toolbox software does so by *softening* each constraint, making a violation mathematically acceptable, though

discouraged. The designer may specify the degree of softness in each case, making selected constraints less likely to be violated than others. See "Optimization Problem" on page 2-5 for the mathematical details.

Briefly, you specify a *tolerance band* for each constraint. If the tolerance band is zero, the constraint is hard (no violation allowed). Increasing the tolerance band softens the constraint.

The tolerance band *is not* a limit on the constraint violation, however. (If it were, you would still have a hard constraint.) You need to view it relative to other constraints.

For example, suppose you have two constraints, one on a temperature and the other on a flow rate. You specify a tolerance band of 2 degrees on the temperature constraint, and 20 kg/s on the flow rate constraint. The Model Predictive Controller assumes that violations of these magnitudes are of *equal concern*, and should be handled accordingly.

Estimating States from Measured Data

At the beginning of each sampling instant the controller estimates the current plant state. Accurate knowledge of the state improves prediction accuracy, which, in turn, improves controller performance.

If all plant states are measured, the state estimation problem is relatively simple and requires consideration of measurement noise effects only. Unfortunately, the internal workings of a typical plant are unmeasured, and the controller must estimate their current values from the available measurements. It also estimate the values of any sustained, unmeasured disturbances.

Model Predictive Control Toolbox software provides a default state estimation strategy, which the designer may customize. For details, see "State Estimation" on page 2-17.

Blocking

In figure Controller State at the *k*th Sampling Instant on page 1-15 (b), $M = 4$ and $P= 9$, and the controller is optimizing the first M moves of the

prediction horizon, after which the manipulated variable remains constant for the remaining $P - M = 5$ sampling instants.

The following figure shows an alternative *blocked* strategy—again with 4 planned moves—in which the first occurs at sampling instant k, the next at $k+2$, the next at $k+4$, and the final at $k+6$. A *block* is one or more successive sampling periods during which the manipulated variable is constant. The *block durations* are the number of sampling periods in each block. In figure Blocking Example with Four Moves on page 1-12 the block durations are 2, 2, 2, and 3. (Their sum must equal P.)

Blocking Example with Four Moves

As for the default (unblocked) mode, only the current move, u_k, actually goes to the plant. Thus, as shown in the figure above, the controller has made a plant adjustment at each sampling instant.

So why use blocking? When $P \gg M$ (as is generally recommended), and all M moves are at the beginning of the horizon, the moves tend to be larger (because all but the final move last just one sampling period). Blocking often leads to smoother adjustments, all other things being equal.

See the subsequent case study examples and the literature for more discussion and MIMO design guidelines.

Blocking

In figure Controller State at the *k*th Sampling Instant on page 1-15 (b),
$M = 4$ and $P = 9$, and the controller is optimizing the first M moves of the
prediction horizon, after which the manipulated variable remains constant for
the remaining $P - M = 5$ sampling instants.

The following figure shows an alternative *blocked* strategy—again with 4
planned moves—in which the first occurs at sampling instant *k*, the next at
k+2, the next at *k*+4, and the final at *k*+6. A *block* is one or more successive
sampling periods during which the manipulated variable is constant. The
block durations are the number of sampling periods in each block. In figure
Blocking Example with Four Moves on page 1-12 the block durations are 2, 2,
2, and 3. (Their sum must equal *P*.)

Blocking Example with Four Moves

As for the default (unblocked) mode, only the current move, u_k, actually goes
to the plant. Thus, as shown in the figure above, the controller has made a
plant adjustment at each sampling instant.

So why use blocking? When $P \gg M$ (as is generally recommended), and all
M moves are at the beginning of the horizon, the moves tend to be larger

(because all but the final move last just one sampling period). Blocking often leads to smoother adjustments, all other things being equal.

See the subsequent case study examples and the literature for more discussion and MIMO design guidelines.

Typical Sampling Instant

Model Predictive Control Toolbox design generates a *discrete-time* controller—one that takes action at regularly spaced, discrete time instants. The *sampling instants* are the times at which the controller acts. The interval separating successive sampling instants is the *sampling period*, Δt (also called the *control interval*). This section provides more details on the events occuring at each sampling instant.
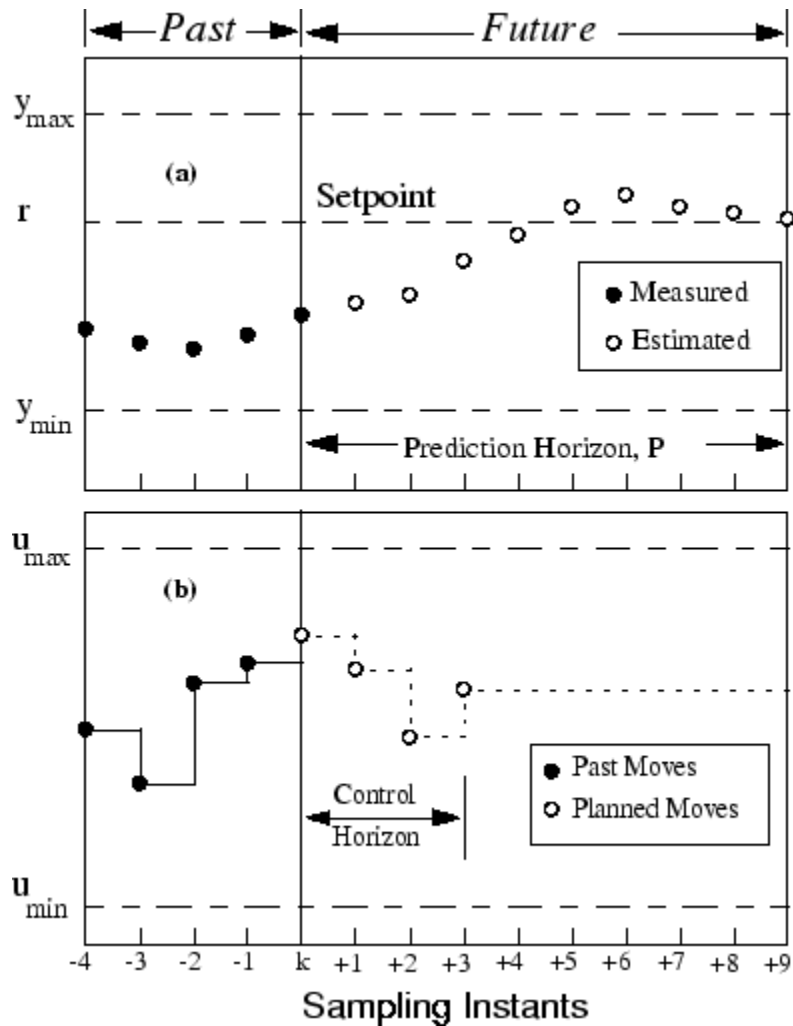
Controller State at the *k*th Sampling Instant

The figure, Controller State at the *k*th Sampling Instant on page 1-15, shows the state of a hypothetical SISO model predictive control system. This system has been operating for many sampling instants. Integer *k* represents the current instant. The latest measured output, y_k, and previous measurements, y_{k-1}, y_{k-2}, ..., are known and are the filled circles in the figure Controller

State at the *k*th Sampling Instant on page 1-15 (a). If there is a measured disturbance, its current and past values would be known (not shown).

Figure Controller State at the *k*th Sampling Instant on page 1-15 (b) shows the controller's previous *moves*, u_{k-4}, ..., u_{k-1}, as filled circles. As is usually the case, a *zero-order hold* receives each move from the controller and holds it until the next sampling instant, causing the step-wise variations shown in figure Controller State at the *k*th Sampling Instant on page 1-15 (b).

To calculate its next *move*, u_k the controller operates in two phases:

1 *Estimation*. In order to make an intelligent move, the controller needs to know the current state. This includes the true value of the controlled variable, $\bar{y}_k$, *and* any internal variables that influence the future trend, $\bar{y}_{k+1},...,\bar{y}_{k+P}$. To accomplish this, the controller uses all past and current measurements and the models $u \rightarrow \bar{y}$, $d \rightarrow \bar{y}$, $v \rightarrow \bar{y}$, and $z \rightarrow y$. For details, see "Prediction" on page 2-22 and "State Estimation" on page 2-17.

2 *Optimization*. Values of setpoints, measured disturbances, and constraints are specified over a finite *horizon* of future sampling instants, *k+1*, *k+2*, ..., *k+P*, where *P* (a finite integer $\geq$ 1) is the *prediction horizon* —see figure Controller State at the *k*th Sampling Instant on page 1-15 (a). The controller computes *M* moves u_k, u_{k+1}, ... u_{k+M-1}, where *M* ($\geq$ 1, $\leq$ *P*) is the *control horizon*—see figure Controller State at the *k*th Sampling Instant on page 1-15 (b). In the hypothetical example shown in the figure, *P* = 9 and *M* = 4. The moves are the solution of a *constrained* optimization problem. For details of the formulation, see "Optimization Problem" on page 2-5.

In the example, the optimal moves are the four open circles in figure Controller State at the *k*th Sampling Instant on page 1-15 (b). The controller predicts that the resulting output values will be the nine open circles in figure Controller State at the *k*th Sampling Instant on page 1-15 (a). Notice that both are within their *constraints*, $u_{min} \leq u_{k+j} \leq u_{max}$ and $y_{min} \leq y_{k+i} \leq y_{max}$.

When it's finished calculating, the controller sends move u_k to the plant. The plant operates with this *constant* input until the next sampling instant, Δt time units later. The controller then obtains new measurements and *totally revises its plan*. This cycle repeats indefinitely.

Reformulation at each sampling instant is essential for good control. The predictions made during the optimization stage are imperfect. Periodic measurement feedback allows the controller to correct for this error and for unexpected disturbances.

Prediction and Control Horizons

You might wonder why the controller bothers to optimize over P future sampling periods and calculate M future moves when it discards all but the first move in each cycle. Indeed, under certain conditions a controller using $P = M = 1$ would be identical to one using $P = M = \infty$. More often, however, the horizon values have an important impact. Some examples follow:

- *Constraints.* Given sufficiently long horizons, the controller can "see" a potential constraint and avoid it—or at least minimize its adverse effects. For example, consider the situation depicted below in which one controller objective is to keep plant output y below an upper bound y_{max}. The current sampling instant is k, and the model predicts the upward trend y_{k+i}. If the controller were looking P_1 steps ahead, it wouldn't be concerned by the constraint until more time had elapsed. If the prediction horizon were P_2, it would begin to take corrective action immediately.
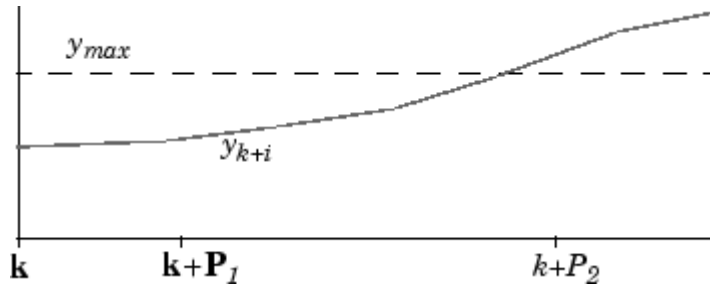
- *Plant delays.* Suppose that the plant includes a pure time delay equivalent to D sampling instants. In other words, the controller's current move, u_k, has no effect until y_{k+D+1}. In this situation it is essential that $P \gg D$ and $M \ll P - D$, as this forces the controller to consider the full effect of each move.

  For example, suppose $D = 5$, $P = 7$, $M = 3$, the current time instant is k, and the three moves to be calculated are u_k, u_{k+1}, and u_{k+2}. Moves u_k, u_{k+1} would have some impact within the prediction horizon, but move u_{k+2} would have none until y_{k+8}, which is outside. Thus, u_{k+2} is indeterminant. Setting $P = 8$ (or $M = 2$) would allow a unique value to be determined. It would be better to increase P even more.

- *Other nonminimum phase plants.* Consider a SISO plant with an inverse-response, i.e., a plant with a short-term response in one direction,

but a longer term response in the opposite direction. The optimization should focus primarily on the longer-term behavior. Otherwise, the controller would move in the wrong direction.

Most designers choose P and M such that controller performance is insensitive to small adjustments in these horizons. Here are typical rules of thumb for a lag-dominant, stable process:

1 Choose the control interval such that the plant's open-loop settling time is approximately 20–30 sampling periods (i.e., the sampling period is approximately one fifth of the dominant time constant).

2 Choose prediction horizon P to be the number of sampling periods used in step 1.

3 Use a relatively small control horizon M, *e.g.*, 3–5.

If performance is poor, you should examine other aspects of the optimization problem and/or check for inaccurate controller predictions.

2

Model Predictive Control Problem Setup

- "Prediction Model" on page 2-2
- "Optimization Problem" on page 2-5
- "Terminal Weights and Constraints" on page 2-12
- "Custom Constraints on Inputs and Outputs" on page 2-15
- "Constraint Softening" on page 2-16
- "State Estimation" on page 2-17
- "QP Matrices" on page 2-22
- "Model Predictive Control Computation" on page 2-28

Prediction Model

The linear model used in Model Predictive Control Toolbox software for prediction and optimization is depicted in the following figure.
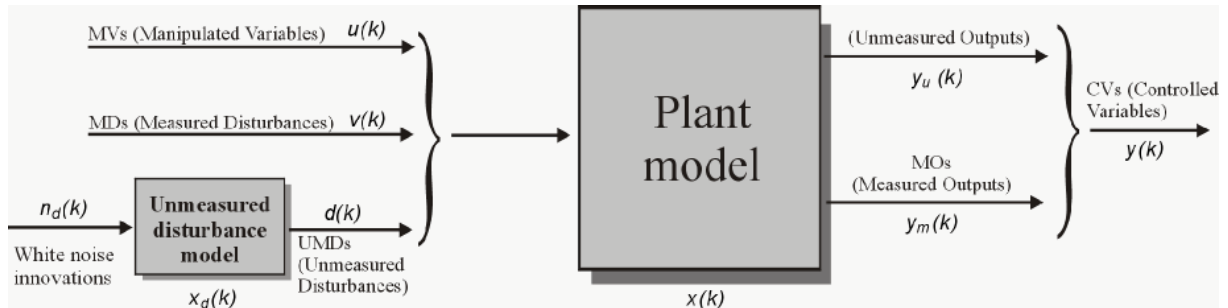
Model Used for Optimization

The model consists of:

- A model of the *plant* to be controlled, whose inputs are the manipulated variables, the measured disturbances, and the unmeasured disturbances

- A model generating the unmeasured *disturbances*

Note When defining a model predictive controller, you must specify a plant model. You do not need to specify a model generating the disturbances, as the controller setup assumes by default that unmeasured disturbances are generated by integrators driven by white noise (see "Output Disturbance Model" on page 2-19 and `setindist`).

The model of the plant is a linear time-invariant system described by the equations

$$x(k+1) = Ax(k) + B_u u(k) + B_v v(k) + B_d d(k)$$
$$y_m(k) = C_m x(k) + D_{vm} v(k) + D_{dm} d(k)$$
$$y_u(k) = C_u x(k) + D_{vu} v(k) + D_{du} d(k)$$

Model Predictive Control Toolbox software accepts plant models specified as LTI objects and identified linear models obtained from input/output data using System Identification Toolbox™, see "Identify Plant from Data".

In the above equations, $d(k)$ collects state disturbances ($B_d \neq 0$) and output disturbances ($D_d \neq 0$).

Note A valid plant model for Model Predictive Control Toolbox software cannot have direct feedthrough of manipulated variables $u(k)$ on the output vector $y(k)$.

The unmeasured disturbance $d(k)$ is modeled as the output of the linear time invariant system:

$$x_d(k+1) = \bar{A}x_d(k) + \bar{B}n_d(k) \tag{2-1}$$

$$d(k) = \bar{C}x_{d(}k) + \bar{D}n_d(k) \tag{2-2}$$

The system described by the above equations is driven by the random Gaussian noise $n_d(k)$, having zero mean and unit covariance matrix. For instance, a step-like unmeasured disturbance is modeled as the output of an integrator. Input disturbance models as in the equations above can be manipulated by using the methods `getindist` and `setindist`.

Note If continuous-time models are supplied, they are internally sampled with the controller's sampling time.

Offsets

In many practical applications, the model matrices *A*, *B*, *C*, *D* are obtained by linearizing a nonlinear dynamical system, such as

$$x' = f(x,u,v,d)$$

$y = h(x, u, v, d)$

at some nominal value $x=x_0$, $u=u_0$, $v=v_0$, $d=d_0$. In these equations x' denotes either the time derivative (continuous time model) or the successor $x(k+1)$ (discrete time model). As an example, x_0, u_0, v_0, d_0 may be obtained by using `findop` on a Simulink® model describing the nonlinear dynamical equations, and A, B, C, D by using `linearize`. The linearized model has the form:

$$
\begin{aligned}
x' &\cong f(x_0,u_0,v_0,d_0) + \nabla_x f(x_0,u_0,v_0,d_0)(x-x_0) + \nabla_x f(x_0,u_0,v_0,d_0)(u-u_0) \\
&\quad + \nabla_v f(x_0,u_0,v_0,d_0)(v-v_0) + \nabla_d f(x_0,u_0,v_0,d_0)(d-d_0) \\
y &\cong h(x_0,u_0,v_0,d_0) + \nabla_x h(x_0,u_0,v_0,d_0)(x-x_0) + \nabla_u h(x_0,u_0,v_0,d_0)(u-u_0) \\
&\quad + \nabla_v h(x_0,u_0,v_0,d_0)(v-v_0) + \nabla_d h(x_0,u_0,v_0,d_0)(d-d_0)
\end{aligned}
$$

The model matrices A, B, C, D are readily obtained from the Jacobian matrices appearing in the equations above.

The linearized dynamics are affected by the constant terms $F=f(x_0, u_0, v_0, d_0)$ and $H=h(x_0, u_0, v_0, d_0)$. For this reason the model predictive control algorithm internally adds a measured disturbance $v=1$, so that F and H can be embedded into B_v and D_v, respectively, as additional columns.

Nonzero offset values d_0 for unmeasured disturbances, while relevant for obtaining the linearized model matrices, are not relevant for the model predictive control problem setup. In fact, only $d-d_0$ can be estimated from output measurements.

Optimization Problem

In this section...
"Standard Form" on page 2-5
"Alternative Cost Function" on page 2-7
"Terminal Weights and Constraints" on page 2-8
"Custom Constraints on Inputs and Outputs" on page 2-11

Standard Form

Assume that estimates of $x(k)$, $x_d(k)$ are available at time k (for state estimation, see "State Estimation" on page 2-17). The model predictive control action at time k is obtained by solving the optimization problem:

$$
\min_{\Delta u(k|k),\dots,\Delta u(m-1+k|k),\varepsilon} \left\{ \sum_{i=0}^{p-1} \left(\sum_{j=1}^{n_y} \left| w_{i+1,j}^y (y_j(k+i+1|k) - r_j(k+i+1)) \right|^2 + \sum_{j=1}^{n_u} \left| w_{i,j}^{\Delta u} \Delta u_j (k+i|k) \right|^2 + \sum_{j=1}^{n_u} \left| w_{i,j}^u (u_j(k+i|k) - u_{j\text{target}}(k+i)) \right|^2 \right) + \rho_\varepsilon \varepsilon^2 \right\}
$$

$$\textbf{(2-3)}$$

In this equation, the subscript "$(\ )_j$" denotes the j component of a vector, "$(k+i\,|\,k)$" denotes the value predicted for time $k+i$. This predictive value is based on the information available at time k. $r(k)$ is the current sample of the output reference, subject to

$$
u_{j\min}(i) - \varepsilon V_{j\min}^u(i) \le u_j\left(k+i|k\right) \le u_{j\max}(i) + \varepsilon V_{j\max}^u(i)
$$

$$
\Delta u_{j\min}(i) - \varepsilon V_{j\min}^{\Delta u}(i) \le \Delta u_j\left(k+i|k\right) \le \Delta u_{j\max}(i) + \varepsilon V_{j\max}^{\Delta u}(i)
$$

$$
y_{j\min}(i) - \varepsilon V_{j\min}^y(i) \le y_j\left(k+i+1|k\right) \le y_{j\max}(i) + \varepsilon V_{j\max}^y(i)
$$

$$
\Delta u\left(k+h|k\right) = 0
$$

$$
\varepsilon \ge 0
$$

where

$i = 0, \dots, p-1$

$h = m, \dots, p-1$

with respect to the sequence of input increments $\{\Delta u(k|k),...,\Delta u(m\text{-}1+k|k)\}$ and to the slack variable ε. The control action sent to the plant is $u(k)=u(k-1)+\Delta u(k|k)^*$. In this case, $\Delta u(k|k)^*$ is the first element of the optimal sequence.

Note Only the measured output vector $y_m(k)$ is fed back to the model predictive controller. However, $r(k)$ is a reference for *all* the outputs (measured and unmeasured).

When the reference r is not known in advance, the current reference $r(k)$ is used over the whole prediction horizon, so $r(k+i+1)=r(k)$ in Equation 2-3.

In model predictive control, the exploitation of future references is referred to as *anticipative action (*or *look-ahead* or *preview*). You can perform a similar anticipative action for measured disturbances $v(k)$. In this prediction, you obtain $d(k+i)$ by setting $n_d(k+i)=0$ as shown in Model Used for Optimization on page 2-2.

$w_{i,j}^{\Delta u}$, $w_{i,j}^{u}$, $w_{i,j}^{y}$ are nonnegative weights for the corresponding variable. The smaller the value of w, the less important the behavior of the corresponding variable is to the overall performance.

$u_{j,min}$ $u_{j,max}$, $\Delta u_{j,min}$, $\Delta u_{j,max}$, $y_{j,min}$, $y_{j,max}$ are lower/upper bounds on the corresponding variables. In Equation 2-3, the constraints on u, Δu, and y are relaxed or *softened* by introducing the slack variable $\varepsilon \geq 0$. In Equation 2-4, the weight ρ_ε on the slack variable ε penalizes the violation of the constraints. As ρ_ε increases relative to the input and output weights, the controller gives minimization of constraint violations higher priority.

The Equal Concern for the Relaxation (ECR) vectors V_{min}^{u}, V_{max}^{u}, $V_{min}^{\Delta u}$, V_{max}^{Du}, V_{min}^{y}, V_{max}^{y} have nonnegative entries that quantify the concern for relaxing the corresponding constraint; the larger V, the *soft*er the constraint. $V=0$ means the constraint is *hard* and cannot be violated. By default, all

input constraints are hard ($V_{\min}^u = V_{\max}^u = V_{\min}^{\Delta u} = V_{\max}^{\Delta u} = 0$) and all output

constraints are soft ($V_{\min}^y = V_{\max}^y = 1$). Also, by default:

$$\rho_e = 10^5 \max\left\{w_{i,j}^{\Delta u}, w_{i,j}^u, w_{i,j}^y\right\}.$$

(2-4)

The model predictive controller penalizes the worst case soft constraint violation—the one for which the inclusion of the non-zero slack variable (and the associated ECR value) allows the constraint to be satisfied at equality. As the model predictive controller attempts to minimize the cost function, it might increase violations of other soft constraints. You can use the ECR values to adjust the priority. Doing so allows you to determine which constraint is selected as the worst-case violation.

Vector $u_{\text{target}}(k+i)$ is a setpoint for the input vector. You typically uses u_{target} when the number of inputs is greater than the number of outputs. Doing so specifies a preferred value for the inputs when all other objectives have been achieved.

As mentioned earlier, only $\Delta u(k\,|\,k)$ is actually used to compute $u(k)$. The remaining samples $\Delta u(k+i\,|\,k)$ are discarded, and a new optimization problem based on $y_m(k+1)$ is solved at the next sampling step $k+1$.

The algorithm implemented in the Model Predictive Control Toolbox software uses different procedures depending on the presence of constraints. If there are no constraints, the controller uses a fast analytical solution to obtain its optimal moves at each sampling instant. Otherwise, a Quadratic Programming (QP) solver is used. The matrices associated with the quadratic optimization problem are described in "QP Matrices" on page 2-22.

If, for numerical reasons, the QP problem becomes infeasible, the second sample from the previous optimal sequence is applied, i.e. $u(k)=u(k–1)+\Delta^* u(k\,|\,k–1)$.

Alternative Cost Function

You have the option to use the following quadratic objective instead of the standard one (Equation 2-3):

$$J(\Delta u, \varepsilon) = \sum_{i=0}^{p-1} \left[y(k+i+1|k) - r(k+i+1) \right]^T Q \left[y(k+i+1|k) - r(k+i+1) \right] + \Delta u(k+i|k)^T R_{\Delta u} (k+i|k)$$

$$+ \left[u(k+i|k) - u_{target}(k+i) \right]^T R_u \left[u(k+i|k) - u_{target}(k+i) \right] + \rho_e \varepsilon^2 \qquad \textbf{(2-5)}$$

In this equation, Q is an n_y by n_y matrix, and $R_{\Delta u}$ and R_u are n_u by n_u matrices, all positive semidefinite. Equation 2-5 allows nonzero, off-diagonal weights but uses the same weights at each step in the prediction horizon.

Equation 2-3 and Equation 2-5 are equivalent when:

- Weights $w_{i,j}^y$, $w_{i,j}^{\Delta u}$, and $w_{i,j}^u$ are constant for all $i = 1, ..., p$.

- Matrices Q, $R_{\Delta u}$ and R_u are diagonal with the squares of the weights $w_{i,j}^y$,

  $w_{i,j}^{\Delta u}$, and $w_{i,j}^u$ respectively as their diagonal elements.

Note When using the alternative cost function, you must define the controller using MATLAB® commands. The Model Predictive Control Toolbox design tool does not provide this option.

Terminal Weights and Constraints

Terminal weights are the quadratic weights *Wy* on *y(t+p)* and *Wu* on *u(t + p –* 1). The variable *p* is the prediction horizon. You apply the quadratic weights at time *k +p* only, such as the prediction horizon's final step. Using terminal weights, you can achieve infinite horizon control that guarantees closed-loop stability. However, before using terminal weights, you must distinguish between problems with and without constraints.

Terminal constraints are the constraints on *y(t + p)* and *u(t + p – 1)*, where *p* is the prediction horizon. You can use terminal constraints as an alternative way to achieve closed-loop stability by defining a terminal region.

Note You can use terminal weights and constraints only at the command-line. See `setterminal`.

For the relatively simple unconstrained case, a terminal weight can make the finite-horizon Model Predictive Controller behave as if its prediction horizon were infinite. For example, the MPC controller behavior is identical to a linear-quadratic regulator (LQR). The standard LQR derives from the cost function:

$$J(u) = \sum_{i=1}^{\infty} x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1)$$

(2-6)

where x is the vector of plant states in the standard state-space form:

$$x(k + 1) = Ax + Bu(k)$$

(2-7)

The LQR provides nominal stability provided matrices Q and R meet certain conditions. You can convert the LQR to a finite-horizon form as follows:

$$J(u) = \sum_{i=1}^{p-1} [x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1)] + x(k+p)^T Q_p x(k+p)$$

(2-8)

where Q_p, the terminal penalty matrix, is the solution of the Riccati equation:

$$Q_p = A^T Q_p A - A^T Q_p B (B^T Q_p B + R)^{-1} B^T Q_p A + Q$$

(2-9)

You can obtain this solution using the `lqr` command in Control System Toolbox software.

In general, Q_p is a full (symmetric) matrix. You cannot use the standard Model Predictive Control Toolbox cost function to implement the LQR cost function. The only exception is for the first $p - 1$ steps if Q and R are diagonal matrices. Also, you cannot use the alternative cost function because it employs identical weights at each step in the horizon. Thus, by definition, the terminal weight differs from those in steps 1 to $p - 1$. Instead, use the following steps:

1 Augment the model (Equation 2-11) to include the weighted terminal states as auxiliary outputs:

$y_{aug}(k) = Q_c x(k)$

where Q_c is the Cholesky factorization of Q_p such that $Q_p = Q_c^T Q_c$.

2 Define the auxiliary outputs y_{aug} as unmeasured, and specify zero weight to them.

3 Specify unity weight on y_{aug} at the last step in the prediction horizon using `setterminal`.

To make the Model Predictive Controller entirely equivalent to the LQR, use a control horizon equal to the prediction horizon. In an unconstrained application, you can use a short horizon and still achieve nominal stability. Thus, the horizon is no longer a parameter to be tuned.

When the application includes constraints, the horizon selection becomes important. The constraints, which are usually softened, represent factors not considered in the LQR cost function. If a constraint becomes active, the control action deviates from the LQR (state feedback) behavior. If this behavior is not handled correctly in the controller design, the controller may destabilize the plant.

For an in-depth discussion of design issues for constrained systems see [2]. Depending on the situation, you might need to include terminal constraints to force the plant states into a defined region at the end of the horizon, after which the LQR can drive the plant signals to their targets. Use `setterminal` to add such constraints to the controller definition.

The standard (finite-horizon) Model Predictive Controller provides comparable performance, if the prediction horizon is long. You must tune the other controller parameters (weights, constraint softening, and control horizon) to achieve this performance.

Tip Robustness to inaccurate model predictions is usually a more important factor than nominal performance in applications.

Related Examples

- Implementing Infinite-Horizon LQR by Setting Terminal Weights in a Finite-Horizon MPC Formulation
- "Providing LQR Performance Using Terminal Penalty" on page 4-70

Custom Constraints on Inputs and Outputs

As discussed previously, Model Predictive Control Toolbox software allows you to add upper and lower bounds on manipulated variables and plant outputs. You can also use the toolbox to constrain linear combinations of these signals. For example, you might want a particular manipulated variable to be greater than the weighted sum of two other manipulated variables. You provide such constraints in the form:

$$Eu(k + j\,|\,k) + Fy(k + j\,|\,k) + Sv(k + j\,|\,k) \leq g + \varepsilon h$$

In this equation, $j = 0,...,p$, E, F, S, g, and h are constants. The variable p is the prediction horizon, ε is the slack variable used for constraint softening (as in Equation 2-3). Thus, each row of E, F, S, g, and h represents a linear constraint to be imposed at each prediction horizon step.

Note Custom constraints can be specified only at the command line. See `setconstraint`.

Related Examples

- MPC Control with Constraints on a Combination of Input and Output Signals
- MPC Control of a Nonlinear Blending Process

Terminal Weights and Constraints

Terminal weights are the quadratic weights *Wy* on *y(t+p)* and *Wu* on *u(t + p −* 1). The variable *p* is the prediction horizon. You apply the quadratic weights at time *k +p* only, such as the prediction horizon's final step. Using terminal weights, you can achieve infinite horizon control that guarantees closed-loop stability. However, before using terminal weights, you must distinguish between problems with and without constraints.

Terminal constraints are the constraints on *y(t + p)* and *u(t + p − 1)*, where *p* is the prediction horizon. You can use terminal constraints as an alternative way to achieve closed-loop stability by defining a terminal region.

Note You can use terminal weights and constraints only at the command-line. See `setterminal`.

For the relatively simple unconstrained case, a terminal weight can make the finite-horizon Model Predictive Controller behave as if its prediction horizon were infinite. For example, the MPC controller behavior is identical to a linear-quadratic regulator (LQR). The standard LQR derives from the cost function:

$$J(u) = \sum_{i=1}^{\infty} x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1)$$

(2-10)

where *x* is the vector of plant states in the standard state-space form:

$$x(k + 1) = Ax + Bu(k)$$

(2-11)

The LQR provides nominal stability provided matrices Q and R meet certain conditions. You can convert the LQR to a finite-horizon form as follows:

$$J(u) = \sum_{i=1}^{p-1} [x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1)] + x(k+p)^T Q_p x(k+p)$$

(2-12)

where Q_p, the terminal penalty matrix, is the solution of the Riccati equation:

$$Q_p = A^T Q_p A - A^T Q_p B (B^T Q_p B + R)^{-1} B^T Q_p A + Q \qquad \textbf{(2-13)}$$

You can obtain this solution using the `lqr` command in Control System Toolbox software.

In general, Q_p is a full (symmetric) matrix. You cannot use the standard Model Predictive Control Toolbox cost function to implement the LQR cost function. The only exception is for the first $p-1$ steps if Q and R are diagonal matrices. Also, you cannot use the alternative cost function because it employs identical weights at each step in the horizon. Thus, by definition, the terminal weight differs from those in steps 1 to $p-1$. Instead, use the following steps:

1 Augment the model (Equation 2-11) to include the weighted terminal states as auxiliary outputs:

$y_{aug}(k) = Q_c x(k)$

where Q_c is the Cholesky factorization of Q_p such that $Q_p = Q_c^T Q_c$.

2 Define the auxiliary outputs y_{aug} as unmeasured, and specify zero weight to them.

3 Specify unity weight on y_{aug} at the last step in the prediction horizon using `setterminal`.

To make the Model Predictive Controller entirely equivalent to the LQR, use a control horizon equal to the prediction horizon. In an unconstrained application, you can use a short horizon and still achieve nominal stability. Thus, the horizon is no longer a parameter to be tuned.

When the application includes constraints, the horizon selection becomes important. The constraints, which are usually softened, represent factors not considered in the LQR cost function. If a constraint becomes active, the control action deviates from the LQR (state feedback) behavior. If this behavior is not handled correctly in the controller design, the controller may destabilize the plant.

For an in-depth discussion of design issues for constrained systems see [2]. Depending on the situation, you might need to include terminal constraints to force the plant states into a defined region at the end of the horizon, after

which the LQR can drive the plant signals to their targets. Use `setterminal` to add such constraints to the controller definition.

The standard (finite-horizon) Model Predictive Controller provides comparable performance, if the prediction horizon is long. You must tune the other controller parameters (weights, constraint softening, and control horizon) to achieve this performance.

Tip Robustness to inaccurate model predictions is usually a more important factor than nominal performance in applications.

Related Examples

- Implementing Infinite-Horizon LQR by Setting Terminal Weights in a Finite-Horizon MPC Formulation
- "Providing LQR Performance Using Terminal Penalty" on page 4-70

Custom Constraints on Inputs and Outputs

As discussed previously, Model Predictive Control Toolbox software allows you to add upper and lower bounds on manipulated variables and plant outputs. You can also use the toolbox to constrain linear combinations of these signals. For example, you might want a particular manipulated variable to be greater than the weighted sum of two other manipulated variables. You provide such constraints in the form:

$$Eu(k + j \mid k) + Fy(k + j \mid k) + Sv(k + j \mid k) \leq g + \varepsilon h$$

In this equation, $j = 0,...,p$, E, F, S, g, and h are constants. The variable p is the prediction horizon, ε is the slack variable used for constraint softening (as in Equation 2-3). Thus, each row of E, F, S, g, and h represents a linear constraint to be imposed at each prediction horizon step.

Note Custom constraints can be specified only at the command line. See `setconstraint`.

Related Examples

- MPC Control with Constraints on a Combination of Input and Output Signals
- MPC Control of a Nonlinear Blending Process

Constraint Softening

A *hard* constraint cannot be violated. Hard constraints are risky, especially for outputs, because the controller will ignore its other objectives in order to satisfy them. Also, the constraints might be impossible to satisfy in certain situations, in which case the calculations are mathematically *infeasible*.

Model Predictive Control Toolbox software allows you to specify *soft* constraints. These can be violated, but you specify a violation tolerance for each (the *relaxation band*). See Equation 2-3, where ε is the slack variable used for constraint softening.

Soft input and output constraints are scalars or vectors. The latter defines a time-varying relaxation band. The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band. Specifying zero indicates a hard constraint.

See Also

- `getconstraint`
- `setconstraint`
- "Define Soft Output Constraints"
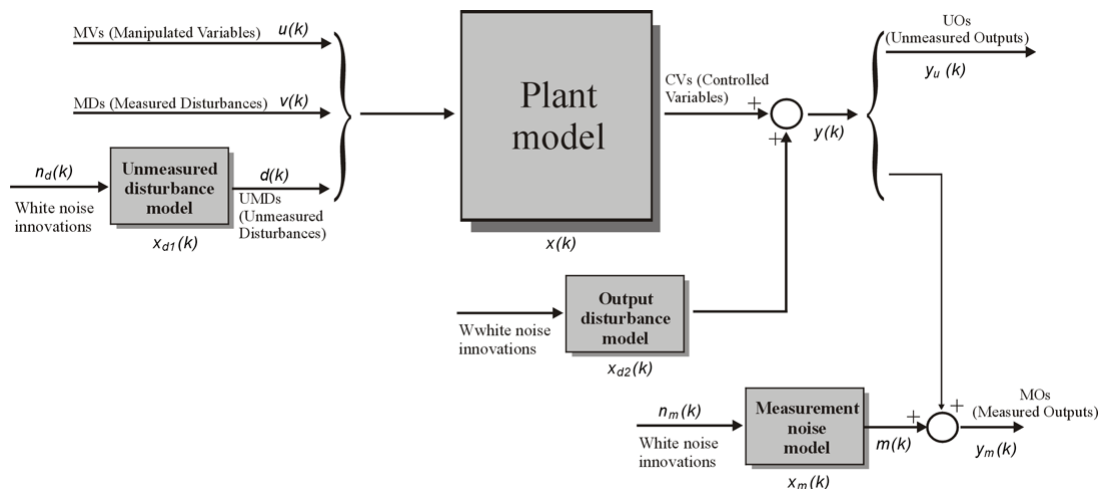
State Estimation

In this section...
"Unmeasured (Input) Disturbance Model" on page 2-17
"Measurement Noise Model" on page 2-19
"Output Disturbance Model" on page 2-19
"State Observer" on page 2-20

As the states $x(k)$, $x_d(k)$ are not directly measurable, predictions are obtained from a state estimator. In order to provide more flexibility, the estimator is based on the model depicted in the following figure.

Model Used for State Estimation

Unmeasured (Input) Disturbance Model

The unmeasured disturbance model is also called the input disturbance model. It allows you to model random disturbances that enter the plant as unmeasured inputs, $d(k)$, which affects the plant states and/or outputs. There is no limit on the number of disturbances contained within $d(k)$, but the combination of plant and disturbance models must be observable. For more information, see "State Observer" on page 2-20.

The $d(k)$ signal is the output of a time-invariant linear system

$$x_d(k+1) = A_d x_d(k) + B_d n_d(k)$$
$$d(k) = C_d x_d(k) + D_d n_d(k)$$

where $n_d(k)$ is a zero-mean, unit variance, random Gaussian input and $x_d(k)$ are the input disturbance model states. You may supply the model in any convenient LTI form (e.g. transfer function). The Model Predictive Control Toolbox will convert it to the state-space model internally.

Delays in your model will be converted to states. This increases the number of states in x_d.

There are three ways to specify the input disturbance model. Suppose `Obj` is an `mpc` object for a plant with two inputs and two outputs. One input is the manipulated variable and the other is an unmeasured disturbance. Let the input disturbance model be an LTI system parameterized by matrices A_d, B_d, C_d, D_d. If x_d is length 2, then C_d must contain 1 row and 2 columns. Use one of the following methods to specify this input disturbance model:

- Set the `Model.Disturbance` property of `Obj`.

  ```
  Obj.Model.Disturbance = ss(Ad,Bd,Cd,Dd);
  ```

- Call `setindist`

  ```
  DistMod = ss(Ad,Bd,Cd,Dd);
  setindist(Obj,'model',DistMod);
  ```

- Use the graphical design tool to import the disturbance model for the `mpc` object. For more information, see "Input Disturbances" on page 5-49.

A default disturbance model is created if the plant model includes a $d(k)$ signal but no input disturbance model is specified. This guarantees asymptotic rejection of plant output disturbances.

Use `getindist` to review the controller's input disturbance model.

Measurement Noise Model

We assume that the measured output vector $y_m(k)$ is corrupted by a measurement noise $m(k)$, which is the output of the linear time-invariant system

$$x_m(k+1) = \tilde{A}x_m(k) + \tilde{B}n_m(k)$$
$$m(k) = \tilde{C}x_m(k) + \tilde{D}n_m(k)$$

The system described by these equations is driven by the random Gaussian noise $n_m(k)$, having zero mean and unit covariance matrix.

Note The objective of the model predictive controller is to bring $y_u(k)$ and $[y_m(k)-m(k)]$ as close as possible to the reference vector $r(k)$. For this reason, the measurement noise model producing $m(k)$ is not needed in the prediction model used for optimization described in "Prediction Model" on page 2-2.

Output Disturbance Model

In order to guarantee asymptotic rejection of output disturbances, the overall model is augmented by an output disturbance model. By default, the output disturbance model is a collection of integrators driven by white noise. Output integrators are added according to the following rule:

1 Measured outputs are ordered by decreasing output weight (in case of time-varying weights, the sum of the absolute values over time is considered for each output channel, and in case of equal output weight the order within the output vector is followed).

2 By following such order, an output integrator is added per measured outputs, unless there is a violation of observability or you force (through the OutputVariables.Integrators property described in "OutputVariables" in the Model Predictive Control Toolbox Reference).

An arbitrary output disturbance model can be specified through the function setoutdist. See also setoutdist for ways to remove the default output integrators.

Use `getoutdist` to obtain the output disturbance model in the form used internally.

State Observer

The state observer is designed to provide estimates of $x(k)$, $x_d(k)$, $x_m(k)$, where $x(k)$ is the state of the plant model, $x_d(k)$ is the overall state of the input and output disturbance model, $x_m(k)$ is the state of the measurement noise model. The estimates are computed from the measured output $y_m(k)$ by the linear state observer

$$
\begin{bmatrix} \hat{x}(k|k) \\ \hat{x}_d(k|k) \\ \hat{x}_m(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{x}_d(k|k-1) \\ \hat{x}_m(k|k-1) \end{bmatrix} + M(y_m(k) - \hat{y}_m(k))
$$

$$
\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{x}_d(k+1|k) \\ \hat{x}_m(k+1|k) \end{bmatrix} = \begin{bmatrix} A\hat{x}(k|k) + B_u u(k) + B_v v(k) + B_d \bar{C} \hat{x}_d(k|k) \\ \bar{A}\hat{x}_d(k|k) \\ \tilde{A}\hat{x}_m(k|k) \end{bmatrix}
$$

$$
\hat{y}_m(k) = C_m \hat{x}(k|k-1) + D_{vm} v(k) + D_{dm}\bar{C}\hat{x}_d(k|k-1) + \tilde{C}\hat{x}_m(k|k-1)
$$

where m denotes the rows of C, D corresponding to measured outputs.

To prevent numerical difficulties in the absence of unmeasured disturbances, the gain M is designed using Kalman filtering techniques (see `kalman` in the Control System Toolbox documentation) on the extended model

$$
\begin{bmatrix} x(k+1) \\ x_d(k+1) \\ x_m(k+1) \end{bmatrix} = \begin{bmatrix} A & B_d\bar{C} & 0 \\ 0 & \bar{A} & 0 \\ 0 & 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \\ x_m(k) \end{bmatrix} + \begin{bmatrix} B_u \\ 0 \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} B_v \\ 0 \\ 0 \end{bmatrix} v(k) + \begin{bmatrix} B_d\bar{D} & 0 & B_u & B_v \\ \bar{B} & 0 & 0 & 0 \\ 0 & \tilde{B} & 0 & 0 \end{bmatrix} \begin{bmatrix} n_d(k) \\ n_m(k) \\ n_u(k) \\ n_v(k) \end{bmatrix}
$$

$$
y_m(k) = \begin{bmatrix} C_m & D_{dm}\bar{C} & \tilde{C} \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \\ x_m(k) \end{bmatrix} + D_{vm}v(k) + \begin{bmatrix} D_{dm}\bar{D} & \tilde{D} & 0 & D_{vm} \end{bmatrix} \begin{bmatrix} n_d(k) \\ n_m(k) \\ n_u(k) \\ n_v(k) \end{bmatrix} \tag{2-14}
$$

where $n_u(k)$ and $n_v(k)$ are additional unmeasured white noise disturbances having unit covariance matrix and zero mean, that are added on the vector of manipulated variables and the vector of measured disturbances, respectively, to ease the solvability of the Kalman filter design.

Note The overall state-space realization of the combined plant and disturbance models must be observable for the state estimation design to succeed. Model Predictive Control Toolbox software first checks for observability of the plant, provided that this is given in state-space form. After all models have been converted to discrete-time, delay-free, state-space form and combined, observability of the overall extended model is checked (see `setestim` and "Construction and Initialization" in the Model Predictive Control Toolbox Reference).

Note also that observability is only checked numerically. Hence, for large models of badly conditioned system matrices, unobservability may be reported by the toolbox even if the system is observable.

See also `getestim` and `setestim` for details on the methods you can use to access and modify properties of the state estimator.

QP Matrices

This section describes the matrices associated with the model predictive control optimization problem described in "Optimization Problem" on page 2-5.

- "Prediction" on page 2-22
- "Optimization Variables" on page 2-23
- "Cost Function" on page 2-25
- "Constraints" on page 2-26

Prediction

Assume that the disturbance model in Equation 2-1 and Equation 2-2 is a unit gain for example, $d(k)=n_d(k)$ is a white Gaussian noise). You can denote this problem as

$$x \leftarrow \begin{bmatrix} x \\ x_d \end{bmatrix}, A \leftarrow \begin{bmatrix} A & B_d\bar{C} \\ 0 & \bar{A} \end{bmatrix}, B_u \leftarrow \begin{bmatrix} B_u \\ 0 \end{bmatrix}, B_v \leftarrow \begin{bmatrix} Bv \\ 0 \end{bmatrix}, B_d \leftarrow \begin{bmatrix} B_d\bar{D} \\ \bar{B} \end{bmatrix} C \leftarrow \begin{bmatrix} C & D_d\bar{C} \end{bmatrix}$$

Then, the prediction model is:

$x(k+1) = Ax(k) + B_u u(k) + B_v v(k) + B_d n_d(k)$

$y(k) = Cx(k) + D_v v(k) + D_d n_d(k)$

Next, consider the problem of predicting the future trajectories of the model performed at time k=0. Set $n_d(i)$=0 for all prediction instants i, and obtain

$$y(i \mid 0) = C\left[A^i x(0) + \sum_{h=0}^{i-1} A^{i-1}\left(B_u\left(u(-1) + \sum_{j=0}^{h} \Delta u(j) \right) + B_v v(h) \right) \right] + D_v v(i)$$

This equation gives the solution

$$\begin{bmatrix} y(1) \\ \cdots \\ y(p) \end{bmatrix} = S_x x(0) + S_{u1} u(-1) + S_u \begin{bmatrix} \Delta u(0) \\ \cdots \\ \Delta u(p-1) \end{bmatrix} + H_v \begin{bmatrix} v(0) \\ \cdots \\ v(p) \end{bmatrix}$$

where

$$S_x = \begin{bmatrix} CA \\ CA^2 \\ \cdots \\ CA^p \end{bmatrix} \in \Re^{pn_y \times n_x}, S_{u1} = \begin{bmatrix} CB_u \\ CB_u + CAB_u \\ \cdots \\ \sum_{h=0}^{p-1} CA^h B_u \end{bmatrix} \in \Re^{pn_y \times n_u}$$

$$S_u = \begin{bmatrix} CB_u & 0 & \cdots & 0 \\ CB_u + CAB_u & CB_u & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{h=0}^{p-1} CA^h B_u & \sum_{h=0}^{p-2} CA^h B_u & \cdots & CB_u \end{bmatrix} \in \Re^{pn_y \times pn_u}$$

$$H_v = \begin{bmatrix} CB_v & D_v & 0 & \cdots & 0 \\ CAB_v & CB_v & D_v & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ CA^{p-1} B_v & CA^{p-2} B_v & CA^{p-3} B_v & \cdots & D_v \end{bmatrix} \in \Re^{pn_y \times (p+1)n_v}.$$

Optimization Variables

Let m be the number of free control moves, and let $z = [z_0; \ldots; z_{m-1}]$. Then,

$$\begin{bmatrix} \Delta u(0) \\ \cdots \\ \Delta u(p-1) \end{bmatrix} = J_M \begin{bmatrix} z_0 \\ \cdots \\ z_{m-1} \end{bmatrix}$$

(2-15)

where J_M depends on the choice of blocking moves. Together with the slack variable ε, vectors $z_0, \ldots, z_{m-1}$ constitute the free optimization variables of the optimization problem. In the case of systems with a single manipulated variables, $z_0, \ldots, z_{m-1}$ are scalars.
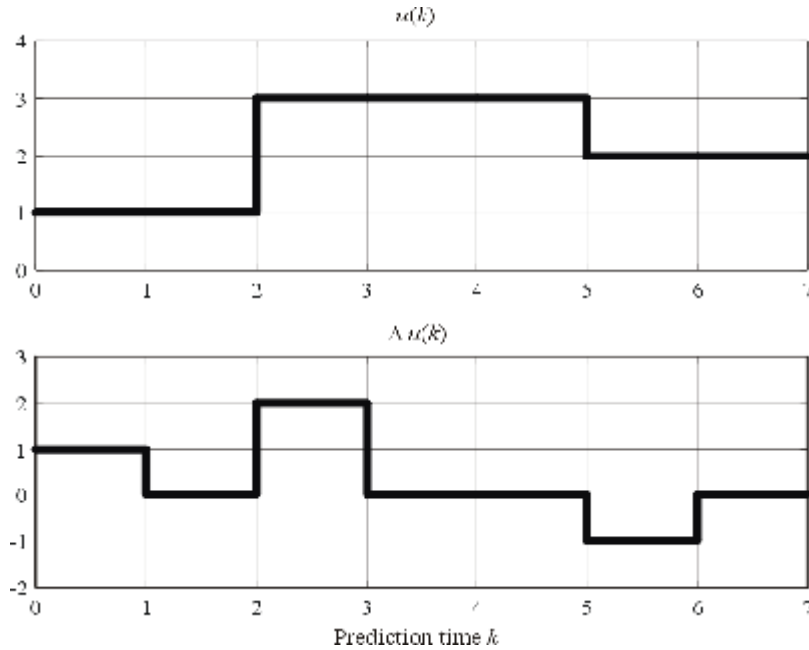
Blocking Moves: Inputs and Input Iincrements for moves=[2 3 2]

Consider the blocking moves depicted in Blocking Moves: Inputs and Input Iincrements for moves=[2 3 2] on page 2-24. This graph corresponds to the choice moves=[2 3 2], or, equivalently, $u(0)=u(1)$, $u(2)=u(3)=u(4)$, $u(5)=u(6)$, $\Delta u(0)=z0$, $\Delta u(2)=z1$, $\Delta u(5)=z2$, $\Delta u(1)=\Delta u(3)=\Delta u(4)=\Delta u(6)=0$.

Then, the corresponding matrix J_M is

$$
J_M = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}
$$

Cost Function

- "Standard Form" on page 2-25
- "Alternative Cost Function" on page 2-26

Standard Form

The function to be optimized is

$$J(z,\varepsilon) = \left(\begin{bmatrix} u(0) \\ \cdots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{target}(0) \\ \cdots \\ u_{target}(p-1) \end{bmatrix}\right)^T W_u^2 \left(\begin{bmatrix} u(0) \\ \cdots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{target}(0) \\ \cdots \\ u_{target}(p-1) \end{bmatrix}\right) + \begin{bmatrix} \Delta u(0) \\ \cdots \\ \Delta u(p-1) \end{bmatrix}^T W_{\Delta u}^2 \begin{bmatrix} \Delta u(0) \\ \cdots \\ \Delta u(p-1) \end{bmatrix}$$

$$+ \left(\begin{bmatrix} y(1) \\ \cdots \\ y(p) \end{bmatrix} - \begin{bmatrix} r(1) \\ \cdots \\ r(p) \end{bmatrix}\right)^T W_y^2 \left(\begin{bmatrix} y(1) \\ \cdots \\ y(p) \end{bmatrix} - \begin{bmatrix} r(1) \\ \cdots \\ r(p) \end{bmatrix}\right) + \rho_\varepsilon \varepsilon^2$$

where

$$W_u = \mathrm{diag}\left(w_{0,1}^u, w_{0,2}^u, ..., w_{0,n_u}^u, ..., w_{p-1,1}^u, w_{p-1,2}^u, ..., w_{p-1,n_u}^u\right)$$

$$W_{\Delta u} = \mathrm{diag}\left(w_{0,1}^{\Delta u}, w_{0,2}^{\Delta u}, ..., w_{0,n_u}^{\Delta u}, ..., w_{p-1,1}^{\Delta u}, w_{p-1,2}^{\Delta u}, ..., w_{p-1,n_u}^{\Delta u}\right)$$

$$W_y = \mathrm{diag}\left(w_{1,1}^y, w_{1,2}^y, ..., w_{1,n_y}^y, ..., w_{p,1}^y, w_{p,2}^y, ..., w_{p,n_y}^y\right) \tag{2-16}$$

Finally, after substituting $u(k)$, $\Delta u(k)$, $y(k)$, $J(z)$ can be rewritten as

$$J(z,\varepsilon) = \rho_\varepsilon \varepsilon^2 + z^T K_{\Delta u} z + 2\left(\begin{bmatrix} r(1) \\ \cdots \\ r(p) \end{bmatrix}^T K_r + \begin{bmatrix} v(0) \\ \cdots \\ v(p) \end{bmatrix}^T K_v + u(-1)^T K_u + \begin{bmatrix} u_{target}(0) \\ \cdots \\ u_{target}(p-1) \end{bmatrix}^T K_{ut} + x(0)^T K_x\right) z$$

$$+ \text{constant} \tag{2-17}$$

Note You may want the QP problem to remain strictly convex. If the condition number of the Hessian matrix $K_{\Delta U}$ is larger than 10^{12}, add the quantity `10*sqrt(eps)` on each diagonal term. You can use this solution only when all input rates are unpenalized ($W^{\Delta u}=0$) (see "Weights" in the Model Predictive Control Toolbox reference documentation).

Alternative Cost Function

If you are using the alternative cost function shown in Equation 2-5, Equation 2-16 is replaced by the following:

$$
\begin{aligned}
W_u &= \text{blkdiag}(R_u, ..., R_u) \\
W_{\Delta u} &= \text{blkdiag}(R_{\Delta u}, ..., R_{\Delta u}) \\
W_y &= \text{blkdiag}(Q, ..., Q)
\end{aligned}
\tag{2-18}
$$

In this case, the block-diagonal matrices repeat p times, for example, once for each step in the prediction horizon.

You also have the option to use a combination of the standard and alternative forms. See "Weights" in the Model Predictive Control Toolbox reference documentation for more details.

Constraints

Next, consider the limits on inputs, input increments, and outputs along with the constraint $\varepsilon \geq 0$.

$$
\begin{bmatrix}
y_{\min}(1) - \varepsilon V^y_{\min}(1) \\
\cdots \\
y_{\min}(p) - \varepsilon V^y_{\min}(p) \\
u_{\min}(0) - \varepsilon V^u_{\min}(0) \\
\cdots \\
u_{\min}(p-1) - \varepsilon V^u_{\min}(p-1) \\
\Delta u_{\min}(0) - \varepsilon V^{\Delta u}_{\min}(0) \\
\cdots \\
\Delta u_{\min}(p-1) - \varepsilon V^{\Delta u}_{\min}(p-1)
\end{bmatrix}
\leq
\begin{bmatrix}
y(1) \\
\cdots \\
y(p) \\
u(0) \\
\cdots \\
u(p-1) \\
\Delta u(0) \\
\cdots \\
\Delta u(p-1)
\end{bmatrix}
\leq
\begin{bmatrix}
y_{\max}(1) + \varepsilon V^y_{\max}(1) \\
\cdots \\
y_{\max}(p) + \varepsilon V^y_{\max}(p) \\
u_{\max}(0) + \varepsilon V^u_{\max}(0) \\
\cdots \\
u_{\max}(p-1) + \varepsilon V^u_{\max}(p-1) \\
\Delta u_{\max}(0) + \varepsilon V^{\Delta u}_{\max}(0) \\
\cdots \\
\Delta u_{\max}(p-1) + \varepsilon V^{\Delta u}_{\max}(p-1)
\end{bmatrix}
$$

Note To reduce computational effort, the controller automatically eliminates extraneous constraints, such as infinite bounds. Thus, the constraint set used in real time may be much smaller than that suggested in this section.

Similar to what you did for the cost function, you can substitute $u(k)$, $\Delta u(k)$, $y(k)$, and obtain

$$M_z z + M_\varepsilon \varepsilon \leq M_{\lim} + M_v \begin{bmatrix} v(0) \\ \cdots \\ v(p) \end{bmatrix} + M_u u(-1) + M_x x(0)$$

(2-19)

In this case, matrices $M_z, M_\varepsilon, M_{\lim}, M_v, M_u, M_x$ are obtained from the upper and lower bounds and ECR values.

Model Predictive Control Computation

This section describes how the model predictive control optimization problem is solved at each time step k (in `mpcmove`, `mpc_sfun.mex`, and `mpcloop_engine.mex`). This solution uses the matrices built at initialization described in "QP Matrices" on page 2-22.

- "Unconstrained Model Predictive Control" on page 2-28
- "Constrained Model Predictive Control" on page 2-28
- "MPC QP Solver" on page 2-28

Unconstrained Model Predictive Control

The optimal solution is computed analytically

$$z^* = -K_{\Delta u}^{-1} \left(\begin{bmatrix} r(1) \\ \cdots \\ r(p) \end{bmatrix}^T K_r + \begin{bmatrix} v(0) \\ \cdots \\ v(p) \end{bmatrix} K_v + u(-1)^T K_u + \begin{bmatrix} u_{target}(0) \\ \cdots \\ u_{target}(p-1) \end{bmatrix}^T K_{ut} + x(0)^T K_x \right)^T$$

and the model predictive controller sets $\Delta u(k) = z^*{}_0$, $u(k) = u(k-1) + \Delta u(k)$.

Constrained Model Predictive Control

The controller computes the optimal solution z^* and ε^* by solving the quadratic program (QP) described in Equation 2-17 and Equation 2-19.

MPC QP Solver

The model predictive controller QP solver converts an MPC optimization problem to the general QP form

$$\underset{x}{Min}(f^T x + \frac{1}{2} x^T H z)$$

such that

$$Ax \le b$$

where $x^T = [z^T\ \varepsilon]$ are the decisions, H is the Hessian matrix, A is a matrix of linear constraint coefficients, and b and f are vectors. The H and A matrices are constants. The controller computes these during initialization and retrieves them from computer memory when needed. It computes the time-varying b and f vectors at the beginning of each control instant.

The toolbox uses the KWIK algorithm [1] to solve the QP problem, which requires the Hessian to be positive definite. In the very first control step, KWIK uses a *cold start*, in which the initial guess is the unconstrained solution described in "Model Predictive Control Computation" on page 2-28. If this x satisfies the constraints, it is the optimal QP solution, x^*, and the algorithm terminates. Otherwise this means that at least one of the linear inequality constraints must be satisfied as an equality. In this case, KWIK uses an efficient, numerically robust strategy to determine the active constraint set satisfying the standard optimality conditions. In the following control steps, KWIK uses a *warm start*. In this case, the active constraint set determined at the previous control step becomes the initial guess for the next.

Although KWIK is robust, you should consider the following:

- One or more linear constraints might be violated slightly due to numerical round-off errors. The toolbox employs a nonadjustable relative tolerance. This tolerance allows a constraint to be violated by 10^{-6} times the magnitude of each term. Such violations are considered normal and do not generate warning messages.

- The toolbox also uses a nonadjustable tolerance when it tests a solution for optimality.

- The search for the active constraint set is an iterative process. If the iterations reach a problem-dependent maximum, the algorithm terminates.

- If your problem includes hard constraints, these constraints might be *infeasible* (impossible to satisfy). If the algorithm detects infeasibility, it terminates immediately.

In the last two situations, with an abnormal outcome to the search, the controller will retain the last successful control output. For more information,

see, the `mpcmove` command. You can detect an abnormal outcome and override the default behavior as you see fit.

References

[1] Schmid, C. and L.T. Biegler. "Quadratic programming methods for reduced hessian SQP." *Computers & Chemical Engineering*. Vol. 18, Number 9, 1994, pp. 817–832.

Model Predictive Control Simulink Library

MPC Library

The MPC Simulink Library provides two blocks you can use to implement MPC control in Simulink, MPC Controller, and Multiple MPC Controllers.

Access the library using the Simulink Library Browser or by typing `mpclib` at the command prompt. The following figure shows the library contents.
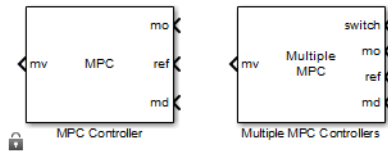
MPC Simulink® Library

Once you have access to the library, you can add one of its blocks to your Simulink model by clicking-and-dragging or copying-and-pasting.

MPC Controller Block

The MPC Controller block represents a single MPC controller. You can adjust plant inputs to control plant outputs, accounting for constraints, including actuator limits.

MPC Controller Block Mask

Insert an MPC Controller block in your Simulink model and then specify its properties. Double-click on the block to open its mask. The following figure shows the mask's default settings. This section shows you how to configure the controller by:

- Specifying required parameters.
- Enabling optional inports and outputs.
- Connecting appropriate signals.
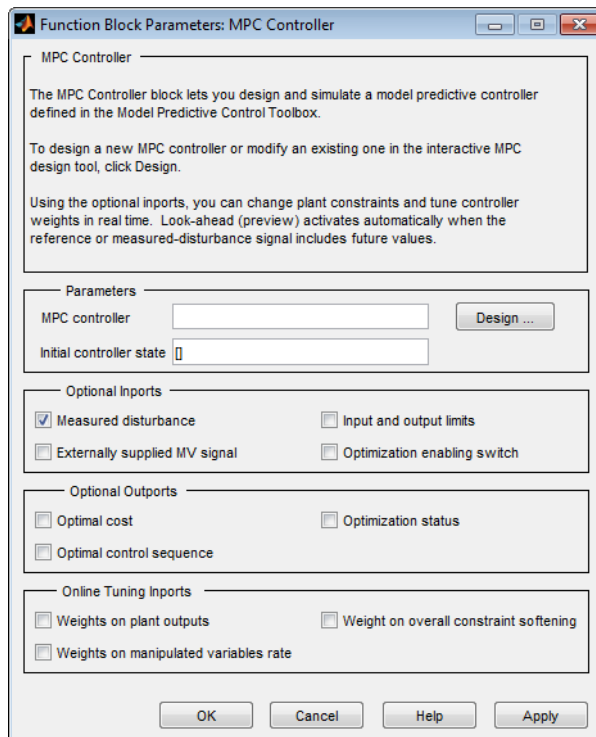
MPC Controller Block Mask

MPC Controller Parameters

Specify the MPC Controller
Specify a valid MPC object in the MPC Controller field in one of two ways:

- Type the name of an MPC object saved in your workspace. To review its settings before running a simulation, click Design to load the named object into the MPC design tool.

- If your installation includes Simulink Control Design™ software, connect the MPC Controller block to the plant it controls. Leaving the MPC controller field empty, click Design. The block constructs a default MPC object by linearizing the plant at the default operating point and exports this defaults controller to the base workspace. If the default operating

point is not the one you want, see "Importing a Plant Model" on page 5-8 for help with importing a new plant model.

Note You can run closed-loop simulations with a linear plant in the design tool, allowing you to tune the controller parameters. To test the controller in Simulink, export it from the design tool to the workspace and run the simulation in Simulink.

Indicate Initial Controller State

If you do not specify an initial controller state, the controller uses a default initial condition in simulations. To change the initial state, specify an mpcstate object. See "MPC Simulation Options Object" in the *Model Predictive Control Toolbox Reference*.

Connect Signals

The MPC Controller requires at least two inputs and generates at least one output. There are also optional inputs and outputs. In most cases, the signals are vectors comprised of plant variables. Verify that the signal dimensions and the sequence of elements within each vector signal are consistent with your controller definition.

Required inputs

Connect the following to the indicated controller inports:

- Measured output variables (mo). Connect the measured plant output variables to the MPC Controller's mo inport as a vector signal, length $n_{ym} \geq 1$. This provides the controller's feedback.

- References (ref). Your controller's prediction model contains $n_y \geq n_{ym}$ output variables. Each must have a reference target or setpoint value. Connect this 1-by-n_y vector signal to the controller's ref inport. This inport defines the reference values at the first step in the controller's prediction horizon. By default, the controller assumes these values hold for the entire horizon.

You can also preview reference signals at run-time. To activate reference previewing, supply the reference input as an $\mathtt{N}\mathtt{x}n_y$ signal, where $1 < N \le p$, and p is the prediction horizon length. The rows 1 to N define the reference values for time instants $\mathtt{t_k}$+1 to $\mathtt{t_k}$+N in the prediction horizon. If N< p, the last row is used for steps $\mathtt{t_k}$+n+1 to $\mathtt{t_k}$+p. See the mpcpreview example.

Required outport

During operation, the controller updates the manipulated variable (mv) outport at each control interval. The mv outport defines adjustments to plant input variables. Connect this vector signal to the plant.

Sample Time

Every $\Delta t (> 0)$ time units, the MPC Controller samples its input signals and updates its output signals. This control interval is defined in the Ts property of the MPC object.

Optional Ports

Feedforward compensation for measured disturbances

If your prediction model includes measured disturbances, ensure that **Measured disturbance** is selected, and click **Apply**. Selecting this parameter adds an inport labeled md. Connect an $\mathtt{N}\ \mathtt{x}\ n_{md}$ signal, where:

- $n_{md} \ge 1$ is the number of measured disturbances coming from the plant

- $N \ge 1$ is the number of sampling instants for which you are supplying measured disturbance values.

If N=1, the previewing is disabled and the signal must contain the vector of measured disturbances at the current controller sampling instant, $\mathtt{t_k}$. N>1 activates previewing. In this case, rows 2 to N must contain estimates of the measured disturbances at future sampling instants $\mathtt{t_k}$+1, ..., $\mathtt{t_k}$+N-1. These estimates allows the controller to preview, or, anticipate future changes in these disturbances and compensate for them at time $\mathtt{t_k}$. See mpcpreview for an illustration of the improved performance this approach can provide.

Externally supplied MV signals

Ideally, the manipulated variable values used in the plant are always identical to those specified by the controller's mv outport. The controller makes this assumption by default. In practice, however, unexpected constraints, disturbances, and plant nonlinearities can prevent these values from being exactly alike. If the actual values of manipulated variables are measured and fed back, the controller's predictions improve. This feature can also smooth the transition between manual and automatic operation. For more information, see "Transfer Bumplessly Between Manual and Automatic Operations" on page 4-41.

Select the **Externally supplied MV signals** check box, and click **Apply** to add an inport labeled ext.mv. Connect the feedback signal, length n_{mv}.

The controller's default behavior is the feedback of its mv outport to the ext.mv inport.

Input and output limits

By default, the controller employs the constraints specified in your controller design as constants. You have the option to update the upper and lower bounds on manipulated and output variables at each controller sampling instant.

Select **Input and output limits**, and click **Apply** to add the following four inports:

- umin defines n_{mv} lower bounds on the manipulated variables (the controller's mv outport).

- umax is a corresponding vector of n_{mv} upper bounds.

- ymin defines n_y lower bounds on the output variables (the same variables to which the ref inport applies).

- ymax is a corresponding vector of n_y upper bounds.

After adding the inports, connect the appropriate signals. See mpcvarbounds for more information.

Disabling optimization

Select this option to control whether or not the block performs its optimization calculations at each sampling instant. For example, if the controller's mv outport is being ignored because the plant is manually controlled, then turning off optimization reduces the computational load. When optimization is off, the mv output is zero.

To activate this option, select **Optimization enabling switch**, and click **Apply**. Selecting this option adds an inport labeled QP switch. Connect a scalar signal. When the signal is zero, optimization occurs. Setting it to nonzero disables optimization.

If you select this option, the Externally supplied MV signals option also activates automatically. To prevent "bumps" when optimization is deactivated temporarily and then reactivates, you must feed the actual MV signal back to the ext.mv inport.

For more information and examples, see "Transfer Bumplessly Between Manual and Automatic Operations" on page 4-41.

Monitoring the optimal cost

This option allows you to monitor the objective function value (optimal cost) obtained by solving the controller's quadratic program (QP). See "Optimization Problem" on page 2-5 for more information.

Select the **Optimal cost** and click **Apply** to add a new scalar outport labeled **cost**.

If the optimization problem is infeasible, that is, one or more hard constraints cannot be satisfied or the solver experiences numerical difficulties. The returned cost is −1.

Monitoring the optimal control sequence

To monitor the controller's planned sequence of future adjustments, select **Optimal control sequence**, and click **Apply**. This option adds an outport labeled mv.seq. At each control instant, the outport contains a $p \times n_{mv}$ matrix signal. The rows correspond to the p steps of the prediction horizon, and the

columns to the n_{mv} manipulated variables. The first row represents current time, k=0. The last row represents time k+p-1.

Monitoring the QP status

Selecting this option allows you to take application-specific actions if the controller's optimization calculation (QP) terminates abnormally. For example, you can set an alarm.

Select **Optimization status**, and click **Apply** to add an outport labeled qp.status. This action generates a scalar signal.

If the QP terminates normally, qp.status (> 0) is the number of QP iterations required. This value is proportional to calculational effort. A large value indicates a difficult QP, and might prevent the controller from making its adjustments in timely fashion.

Possible abnormal terminations are:

- **= 0**: The QP could not be solved in the maximum number of iterations specified in the controller definition.

- **= -1**: The QP solver detected an infeasible QP, that is, it was impossible to satisfy all the hard constraints imposed in the controller definition.

- **= -2**: The QP solver encountered severe numerical difficulties, such as a poorly conditioned QP.

If the QP terminates abnormally, the controller's mv outport contains the most recent successful solution.

Online tuning

Three related options allow you to adjust controller performance during operation. To activate one, select the appropriate check box, and click **Apply**. This adds a new inport with the functionality:

Weights on plant outputs

The added inport is labeled y.wt. Connect a vector signal, length n_y, which defines the nonnegative real weight on reference tracking for each of the n_y output variables. This signal overrides the controller's

MPCobj.Weights.OV property. A larger weight increases the importance of accurate tracking for the corresponding output variable.

Weights on manipulated variables rate
The added inport is labeled du.wt. Connect a vector signal, length n_{mv}, which defines the nonnegative real weight on the adjustment (increment) for each of the n_{mv} manipulated variables. This signal overrides the controller's MPCobj.Weights.MVRate property. A larger weight discourages the controller from making large-magnitude adjustments in the corresponding plant variable.

Weight on overall constraint softening
The added inport is labeled ECR.wt. Connect a scalar nonnegative real signal specifying the weight on the slack variable used to soften constraints. This signal overrides the controller's MPCobj.Weights.ECR property. A larger weight makes all soft constraints harder.

Input Signals

You must connect appropriate Simulink signals to the MPC Controller block's inports. The measured output (mo) and reference (ref) inports are required. You can add optional inports by selecting check boxes at the bottom of the mask.

As shown in the figure, MPC Controller Block Mask on page 3-4 , **Enable measured disturbances** is a default selection and the corresponding inport (md) appears in figure MPC Simulink® Library on page 3-2. This provides feedforward compensation for measured disturbances.

Enable externally supplied MV signals allows you to keep the controller informed of the *actual* manipulated variable values. Ideally, the actual manipulated variables are those specified by the controller block output mv. In practice, unexpected constraints, disturbances, or plant nonlinearities can modify the values actually implemented in the plant. If the actual values are known and fed back to the controller, its predictions improve. This feature can also improve the transition between manual and automatic operation. See "Transfer Bumplessly Between Manual and Automatic Operations" on page 4-41.

Enable input and output limits allows you to specify constraints that vary during a simulation. Otherwise, the block uses the constant constraint values

stored within its MPC Controller object. The example `mpcvarbounds` shows how this option works. It enables inports for lower and upper bounds on the manipulated variables (inports `umin` and `umax`) and lower and upper bounds on the controlled outputs (inports `ymin` and `ymax`). An unconnected constraint inport causes the corresponding variable to be unconstrained.

Enable optimization switch allows you to control whether or not the block performs its optimization calculations at each sampling instant during a simulation. If the controller is output is being ignored during the simulation, e.g., due to a switch to manual control, turning off the optimization reduces the computational load. When optimization is off, the controller output is zero. To turn the optimization off, set the switch input signal to a nonzero value. When the switch input is zero or disconnected, the optimization occurs and the controller output varies in the normal way.

If you select the switching option, the **Enable externally supplied MV signals** option must also be activated. See "Transfer Bumplessly Between Manual and Automatic Operations" on page 4-41 for an example application.

Output Signals

The block updates its output(s) at regular intervals. The MPC object named in the block's **MPC controller** field contains the control interval (its `Ts` property). The object also specifies the number of manipulated variables, n_u, to be calculated and sent to the plant at each control instant. The default outport (labeled `mv`) provides these as a vector signal of dimension n_u.

There are two optional outputs:

• The **Enable optimal cost outport** option adds an outport labeled cost, which contains the value of the optimal objective function obtained when calculating the manipulated variables. See "Optimization Problem" on page 2-5 for the various forms this can take. If the optimization problem is infeasible, i.e., some constraints can't be satisfied or the solver experiences numerical difficulties, the returned cost is -1.

• The **Enable control sequence outport** option creates a new outport labeled `mv.seq`. At each control instant, this outport contains the calculated optimal manipulated variable sequence for the prediction horizon specified in the MPC object.

For more information, see the MPC Controller block reference page.

Look Ahead and Signals from the Workspace

The mask's **Input signals** section allows you to define the reference and/or measured disturbance signals as variables in the workspace. In this case, the block ignores the signals connected to its corresponding inports.

You must create such a signal as a MATLAB structure with two fields: `time` and `signals`. The Simulink **From Workspace** and **To Workspace** blocks use the same format.

For example, to specify a sinusoidal reference signal *sin*(*t*) over a time horizon of 10 seconds, use the following MATLAB commands:

```
time=(0:Ts:10);
ref.time=time;
ref.signals.values=sin(time);
```

where `Ts` is the controller sampling period. After the variable is created, select the **Use custom reference signal** check box and enter the variable name in the edit box.

An alternative would be to run a Simulink simulation in which you connect an appropriate block (**Sine**, in the above example) to a **To Workspace** block.

The **Look ahead** check box enables an anticipative action on the corresponding signal. This option becomes available when you define reference and measured disturbance signals in the workspace. For example, if you define the reference signal as described above, the **Look ahead** option becomes available. Selecting it causes the controller to compensate for the known future reference variations, which usually improves setpoint tracking. When **Look ahead** is disabled or unselected, the controller assumes that the current reference (or measured disturbance) value applies throughout its prediction horizon.

See the `mpcpreview` example for an illustrative example of enabling preview and reading signals from the workspace.

Initialization

If **Initial controller state** is unspecified, as in MPC Controller Block Mask on page 3-4, the controller uses a default initial condition in simulations. You can change the initial condition by specifying an mpcstate object. See "MPC Simulation Options Object" in the Model Predictive Control Toolbox Reference.

Generate Code and Deploy Controller to Real-Time Targets

After designing a controller in Simulink software using the MPC Controller block, you can generate code and deploy it for real-time control. You can deploy the controller to all targets supported by the following products:

- Simulink Coder™
- Embedded Coder®
- xPC Target™

The sampling rate that a controller can achieve in real-time environment is system dependent. For example, for a typical small MIMO control application running on xPC Target, the sampling rate may go as low as 1–10ms. To determine the sampling rate, first test a less aggressive controller whose sampling rate produces acceptable performance on the target. Next, increase the sampling rate and monitor the execution time used by the controller. You can further decrease the sampling rate as long as the optimization safely completes within each sampling period under the normal plant operations.

Note The MPC Controller block is implemented using the MATLAB Function block. To see the structure, right-click the block and select **Mask > Look Under Mask**. Open the MPC subsystem underneath.

Multiple MPC Controllers Block

In this section...
"Limitations" on page 3-15
"Examples" on page 3-15

The Multiple MPC Controllers block allows you to achieve better control of a nonlinear plant over a range of operating conditions.

A controller that works well initially can degrade if the plant is nonlinear and its operating point changes. In conventional feedback control, you might compensate for this degradation by gain scheduling.

In a similar manner, the Multiple MPC Controllers block allows you to transition between multiple MPC controllers in real time in a preordained manner. You design each controller to work well in a particular region of the operating space. When the plant moves away from this region, you instruct another MPC controller to take over.

Limitations

The Multiple MPC Controllers block does not provide all the optional features found in the MPC Controller block. The following ports are currently not available:

- Optional outports such as optimal cost, optimal control sequence, and optimization status
- Optional inports for online tuning

Examples

See the `mpcswitching` and `mpccstr` examples for applications of the Multiple MPC Controllers block.

Relationship of Multiple MPC Controllers to MPC Controller Block

The key difference between the Multiple MPC Controllers and the MPC Controller block is the way in which you designate the controllers to be used.

Listing the controllers

You must provide an ordered list containing *N* names, where *N* is the number of controllers and each name designates a valid MPC object in your base workspace. Each named controller must use the identical set of plant signals (for example, the same measured outputs and manipulated variables). See the Multiple MPC Controllers reference for more information on creating lists.

Designing the controllers

Use your knowledge of the process to identify distinct operating regions and design a controller for each. One convenient approach is to use the Simulink Control Design product to calculate each nominal operating point (typically a steady-state condition). Then, obtain a linear prediction model at this condition. To learn more, see the Simulink Control Design documentation. You must have Simulink Control Design product license to use this approach.

After the prediction models have been defined for each operating region, design each corresponding MPC Controller and give it a unique name in your base workspace.

Defining controller switching

Next, define the switching mechanism that will select among the controllers in real time. Add this mechanism to your Simulink model. For example, you could use one or more selected plant measurements to determine when each controller becomes active.

Your mechanism must define a scalar switching signal in the range 1 to *N*, where *N* is the number of controllers in your list. Connect this signal to the block's switch inport. Set it to 1 when you want the first controller in your list to become active, to 2 when the second is to become active, and so on.

Note The Multiple MPC Controllers block automatically rounds the switching signal to the nearest integer. If the signal is outside the range 1 to *N*, none of the controllers activates and the block output is zero.

Improving prediction accuracy

During operation, all inactive controllers receive the current manipulated variable and measured output signals so they can update their state estimates. These updates minimizes bumps during controller transitions. See "Transfer Bumplessly Between Manual and Automatic Operations" on page 4-41 for more information. It is good practice to enable the **Externally supplied MV signal** option and feedback the actual manipulated variables measured in the plant to the ext.mv inport. This signal is provided to all the controllers in the block's list.

4

Case-Study Examples

Servomechanism Controller

System Model

A position servomechanism consists of a DC motor, gearbox, elastic shaft, and a load.
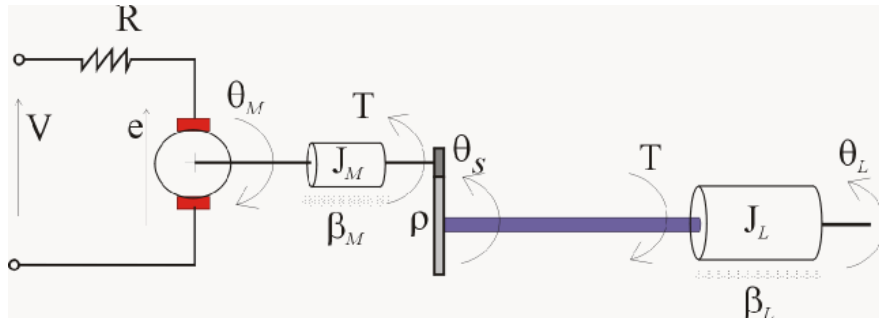
Position Servomechanism Schematic

The differential equations representing this system are

$$\dot{\omega}_L = -\frac{k_\theta}{J_L}\left(\theta_L - \frac{\theta_M}{\rho}\right) - \frac{\beta_L}{J_L}\omega_L$$

$$\dot{\omega}_M = \frac{k_T}{J_M}\left(\frac{V - k_T\omega_M}{R}\right) - \frac{\beta_M\omega_M}{J_M} + \frac{k_\theta}{\rho J_M}\left(\theta_L - \frac{\theta_M}{\rho}\right)$$

where V is the applied voltage, T is the torque acting on the load, $\omega_L = \dot{\theta}_L$ is the load's angular velocity, $\omega_M = \dot{\theta}_M$ is the motor shaft's angular velocity, and the other symbols represent constant parameters (see Parameters Used in the Servomechanism Model on page 4-3 for more information on these).

If you define the state variables as $x_p = \begin{bmatrix} \theta_L & \omega_L & \theta_M & \omega_< \end{bmatrix}^T$, then you can convert the above model to an LTI state-space form:

$$\dot{x}_p = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{k_\theta}{J_L} & \dfrac{\beta_L}{J_L} & \dfrac{k_\theta}{\rho J_L} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{k_\theta}{\rho J_M} & 0 & -\dfrac{k_\theta}{\rho^2 J_M} & \dfrac{\beta_M + \dfrac{k_T^2}{R}}{J_M} \end{bmatrix} x_p + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dfrac{k_T}{R J_M} \end{bmatrix} V$$

$$\theta_L = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x_p$$

$$T = \begin{bmatrix} k_\theta & 0 & \dfrac{k_\theta}{\rho} & 0 \end{bmatrix} x_p$$

Parameters Used in the Servomechanism Model

Symbol	Value (SI Units)	Definition
k_θ	1280.2	Torsional rigidity
k_T	10	Motor constant
J_M	0.5	Motor inertia
J_L	$50 J_M$	Load inertia
ρ	20	Gear ratio
β_M	0.1	Motor viscous friction coefficient
β_L	25	Load viscous friction coefficient
R	20	Armature resistance

Control Objectives and Constraints

The controller must set the load's angular position, θ_L, at a desired value by adjusting the applied voltage, *V*. The only measurement available for feedback is θ_L.

The elastic shaft has a finite shear strength, so the torque, *T*, must stay within specified limits

$$|T| \leq 78.5\text{Nm}$$

Also, the applied voltage must stay within the range

$$|V| \leq 220\text{V}$$

From an input/output viewpoint, the plant has a single input, *V*, which is manipulated by the controller. It has two outputs, one measured and fed back to the controller, θ_L, and one unmeasured, *T*.

The specifications require a fast servo response despite constraints on a plant input and a plant output.

Defining the Plant Model

The first step in a design is to define the plant model.

```
% DC-motor with elastic shaft
%
%Parameters (MKS)
%-------------------------------------------------------
Lshaft=1.0;          %Shaft length
dshaft=0.02;         %Shaft diameter
shaftrho=7850;       %Shaft specific weight (Carbon steel)
G=81500*1e6;         %Modulus of rigidity
tauam=50*1e6;        %Shear strength
Mmotor=100;          %Rotor mass
Rmotor=.1;           %Rotor radius
Jmotor=.5*Mmotor*Rmotor^2;     %Rotor axial moment of inertia
Bmotor=0.1;          %Rotor viscous friction coefficient (A CASO)
R=20;                %Resistance of armature
Kt=10;               %Motor constant
```

```
gear=20;              %Gear ratio
Jload=50*Jmotor;      %Load inertia
Bload=25;             %Load viscous friction coefficient
Ip=pi/32*dshaft^4;                  %Polar momentum of shaft (circular) section
Kth=G*Ip/Lshaft;                    %Torsional rigidity (Torque/angle)
Vshaft=pi*(dshaft^2)/4*Lshaft;      %Shaft volume
Mshaft=shaftrho*Vshaft;             %Shaft mass
Jshaft=Mshaft*.5*(dshaft^2/4);      %Shaft moment of inertia
JM=Jmotor;
JL=Jload+Jshaft;
Vmax=tauam*pi*dshaft^3/16;    %Maximum admissible torque
Vmin=-Vmax;

%Input/State/Output continuous time form
%--------------------------------------------------------
AA=[0             1             0             0;
    -Kth/JL       -Bload/JL     Kth/(gear*JL) 0;
    0             0             0             1;
    Kth/(JM*gear) 0             -Kth/(JM*gear^2) -(Bmotor+Kt^2/R)/JM];

BB=[0;0;0;Kt/(R*JM)];
Hyd=[1 0 0 0];
Hvd=[Kth 0 -Kth/gear 0];
Dyd=0;
Dvd=0;

% Define the LTI state-space model
sys=ss(AA,BB,[Hyd;Hvd],[Dyd;Dvd]);
```

Controller Design Using MPCTOOL

The servomechanism model is linear, so you can use the Model Predictive
Control Toolbox design tool (`mpctool`) to configure a controller and test it.

Note To follow this example on your own system, first create the
servomechanism model as explained in "Servomechanism Controller" on page
4-2. This defines the variable `sys` in your MATLAB workspace.

Opening MPCTOOL and Importing a Model

To begin, open the design tool by typing the following at the MATLAB prompt:

```
mpctool
```

Once the design tool has appeared, click the **Import Plant** button. The Plant Model Importer dialog box appears (see the following figure).

By default, the **Import from** option buttons are set to import from the MATLAB workspace, and the box at the upper right lists all LTI models defined there. In the following figure, sys is the only available model, and it is selected. The **Properties** area lists the selected model's key attributes.
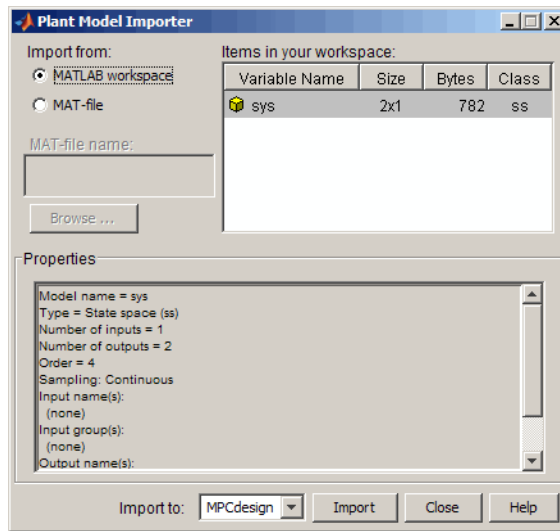
Import Dialog Box with the Servomechanism Model Selected

Make sure your servomechanism model, sys, is selected. Then click the **Import** button. You won't be importing more models, so close the import dialog box.

Meanwhile, the model has loaded, and tables now appear in the design tool's main window (see the figure below). Note the previous diagram enumerates the model's input and output signals.
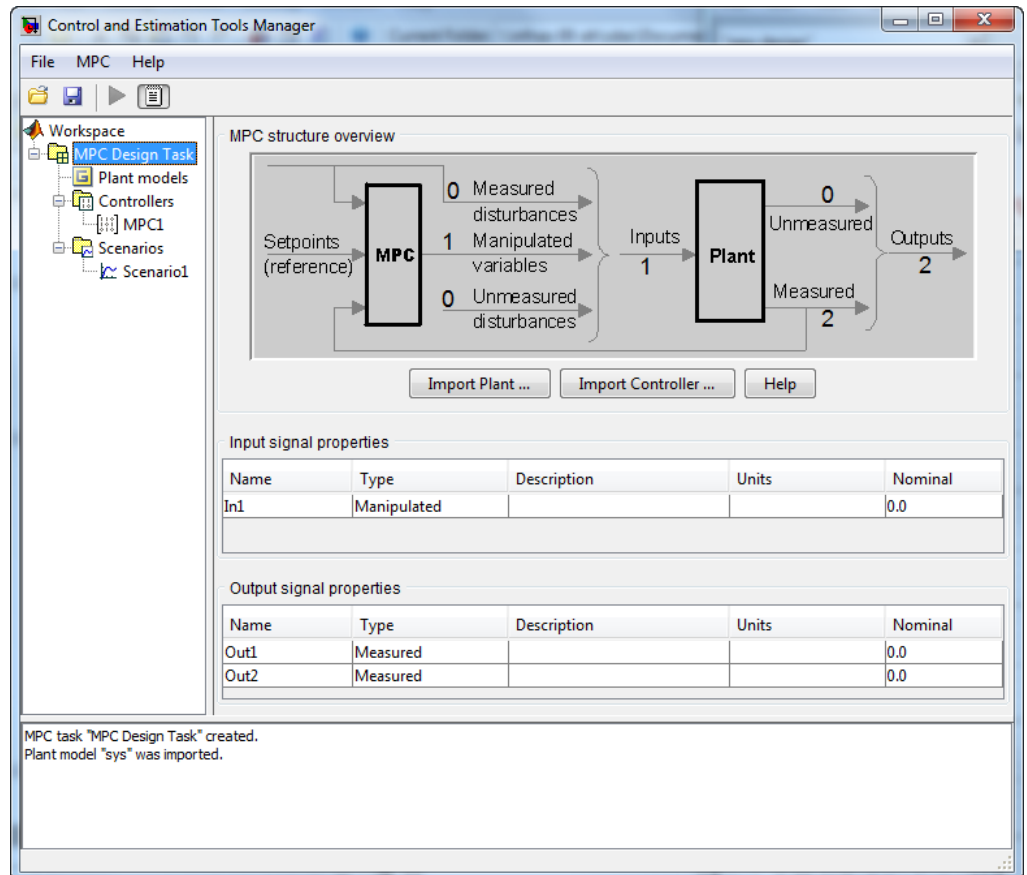
Design Tool After Importing the Plant Model

Specifying Signal Properties

It's essential to specify *signal types* before going on. By default, the design tool assumes all plant inputs are manipulated, which is correct in this case. But it also assumes all outputs are measured, which is not. Specify that the second output is unmeasured by clicking on the appropriate table cell and selecting the **Unmeasured** option.

You also have the option to change the default signal names (`In1`, `Out1`, `Out2`) to something more meaningful (e.g., `V`, `ThetaL`, `T`), enter descriptive

information in the blank **Description** and **Units** columns, and specify a nominal initial value for each signal (the default is zero).

After you've entered all your changes, the design tool resembles the following figure.

Input signal properties

Name	Type	Description	Units	Nominal
V	Manipulated	Applied Voltage	V	0.0

Output signal properties

Name	Type	Description	Units	Nominal
ThetaL	Measured	Angular position	Radians	0.0
T	Measured	Torque applied to load	Nm	0.0

Design Tool After Specifying Signal Properties

Navigation Using the Tree View

Now consider the design tool's left-hand frame. This *tree* is an ordered arrangement of *nodes*. Selecting (clicking) a node causes the corresponding view to appear in the right-hand frame. When the design tool starts, it creates a *root* node named **MPC Design Task** and selects it, as in Design Tool After Importing the Plant Model on page 4-7.

The **Plant models** node is next in the hierarchy. Click on it to list the plant models being used in your design. (Each model name is editable.) The middle section displays the selected model's properties. There is also a space to enter notes describing the model's special features. Buttons allow you to import a new model or delete one you no longer need.

The next node is **Controllers**. You might see a + sign to its left, indicating that it contains subnodes. If so, click on the + sign to expand the tree (as shown in Design Tool After Importing the Plant Model on page 4-7). All the controllers in your design will appear here. By default, you have one: **MPC1**. In general, you might opt to design and test several alternatives.

Select **Controllers** to see a list of all controllers, similar to the **Plant models** view. The table columns show important controller settings: the plant model

being used, the controller sampling period, and the prediction and control horizons. All are editable. For now, leave them at their default values.

The buttons on the **Controllers** view allow you to:

- **Import** a controller designed previously and stored either in your workspace or in a MAT-file.

- **Export** the selected controller to your workspace.

- Create a **New** controller, which will be initialized to the Model Predictive Control Toolbox defaults.

- **Copy** the selected controller to create a duplicate that you can modify.

- **Delete** the selected controller.

Specifying Controller Properties

Select the **MPC1** subnode. The main pane should change to the controller design.

If the selected **Prediction model** is continuous-time, as in this example, the **Control interval** (sampling period) defaults to 1. You need to change this to an application-appropriate value. Set it to 0.1 seconds (as shown in Controller Design View, Models and Horizons Pane on page 4-10). Leave the other values at their defaults for now.
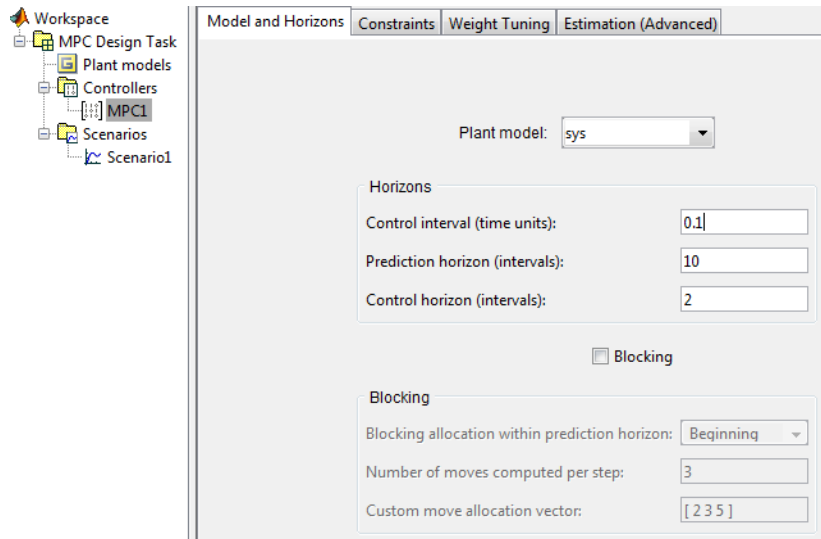
Controller Design View, Models and Horizons Pane

Specifying Constraints

Next, click the **Constraints** tab. The view shown in Controller Design View, Constraints Pane on page 4-10 appears. Enter the appropriate constraint values. Leaving a field blank implies that there is no constraint.
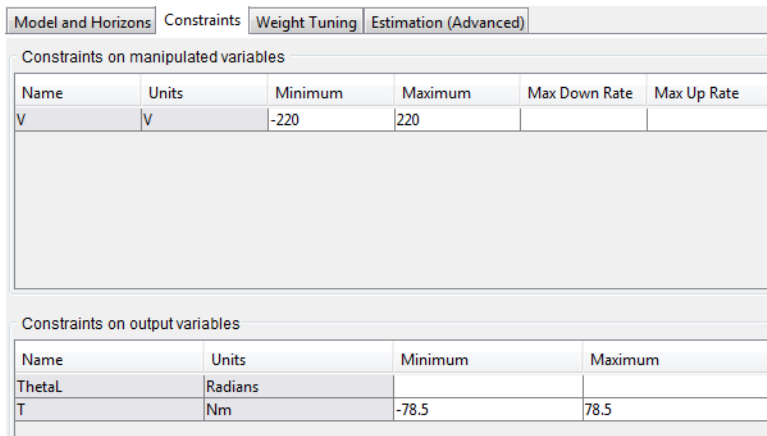
Controller Design View, Constraints Pane

In general, it's good practice to include all known manipulated variable constraints, but it's unwise to enter constraints on outputs unless they are an essential aspect of your application. The limit on applied torque is such a constraint, as are the limits on applied voltage. The angular position has physical limits but the controller shouldn't attempt to enforce them, so you should leave the corresponding fields blank (see Controller Design View, Constraints Pane on page 4-10).

The **Max down rate** should be nonpositive (or blank). It limits the amount a manipulated variable can decrease in a single control interval. Similarly, the **Max up rate** should be nonnegative. It limits the increasing rate. Leave both unconstrained (i.e., blank).

The shaded columns can't be edited. If you want to change this descriptive information, select the root node view and edit its tables. Such changes apply to all controllers in the design.

Weight Tuning

Next, click the **Weight Tuning** tab.

The *weights* specify trade-offs in the controller design. First consider the **Output weights**. The controller will try to minimize the deviation of each output from its *setpoint* or *reference* value. For each sampling instant in the prediction horizon, the controller multiplies predicted deviations for each output by the output's weight, squares the result, and sums over all sampling instants and all outputs. One of the controller's objectives is to minimize this sum, *i.e.*, to provide good *setpoint tracking*. (See "Optimization Problem" on page 2-5 for more details.)

Here, the angular position should track its setpoint, but the applied torque can vary, provided that it stays within the specified constraints. Therefore, set the torque's weight to zero, which tells the controller that setpoint tracking is unnecessary for this output.

Similarly, it's acceptable for the applied voltage to deviate from nominal (it must in order to change the angular position!). Its weight should be zero (the default for manipulated variables). On the other hand, it's probably undesirable for the controller to make drastic changes in the applied voltage. The **Rate weight** penalizes such changes. Use the default, 0.1.

When setting the rates, the relative magnitudes are more important than the absolute values, and you must account for differences in the measurement scales of each variable. For example, if a deviation of 0.1 units in variable A is just as important as a deviation of 100 units in variable B, variable A's weight must be 1000 times larger than that for variable B.
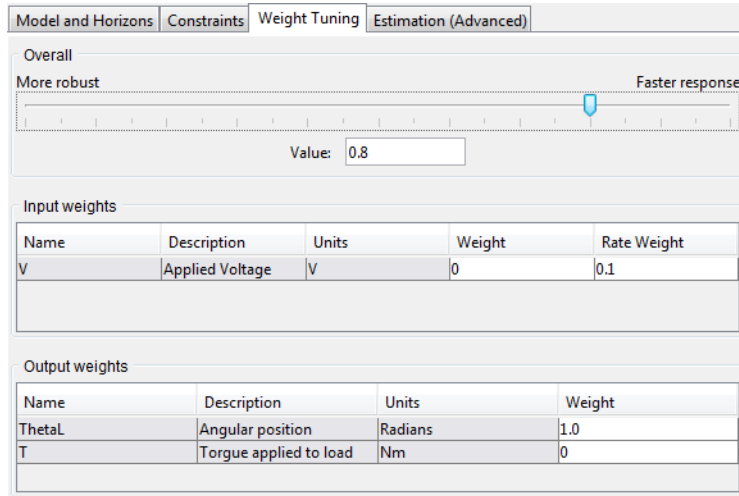
Controller Design View, Weight Tuning Pane

The tables allow you to weight individual variables. The slider at the top adjusts an overall trade-off between controller aggressiveness and setpoint tracking. Moving the slider to the left places a larger overall penalty on manipulated variable changes, making them smaller. This usually increases controller robustness, but setpoint tracking becomes more sluggish.

The **Estimation (Advanced)** tab allows you to adjust the controller's response to unmeasured disturbances (not used in this example).

Defining a Simulation Scenario

If you haven't already done so, expand the **Scenarios** node to show the **Scenario1** subnode (see Design Tool After Importing the Plant Model on page 4-7). Select **Scenario1**.

A *scenario* is a set of simulation conditions. As shown in Simulation Settings View for "Scenario1" on page 4-13, you choose the controller to be used (from among controllers in your design), the model to act as the plant, and the simulation duration. You must also specify all setpoints and disturbance inputs.

Duplicate the settings shown in Simulation Settings View for "Scenario1" on page 4-13, which will test the controller's servo response to a unit-step change in the angular position setpoint. All other inputs are being held constant at their nominal values.

Simulation settings

Controller	MPC1	▼	☑ Close loops	
Plant	sys	▼	☑ Enforce constraints	
Duration	30		Control interval	0.1

Setpoints

Name	Units	Type	Initial Value	Size	Time	Period	Look Ahead
ThetaL	Radians	Step	0.0	1	1.0		☐
T	N m	Constant	0.0				☐

Unmeasured disturbances

Name	Units	Type	Initial Value	Size	Time	Period
ThetaL	Radians	Constant	0.0			
V	Volts	Constant	0.0			

[Simulate] [Help] [Tuning Advisor]

Simulation Settings View for "Scenario1"

Note The **ThetaL** and **V** unmeasured disturbances allow you to simulate additive disturbances to these variables. By default, these disturbances are turned off, i.e., zero.

The **Look ahead** option designates that all future setpoint variations are known. In that case, the controller can adjust the manipulated variable(s) in advance to improve setpoint tracking. This would be unusual in practice, and is not being used here.

Running a Simulation

Once you're ready to run the scenario, click the **Simulate** button or the green arrow on the toolbar.

Note The green arrow tool is available from any view once you've defined at least one scenario. It runs the *active scenario*, i.e., the one most recently selected or modified.

We obtain the results shown in Response to Unit Step in the Angular Position Setpoint on page 4-14. The blue curves are the output signals, and the gray curves are the corresponding setpoints. The response is very sluggish, and hasn't settled within the 30-second simulation period.
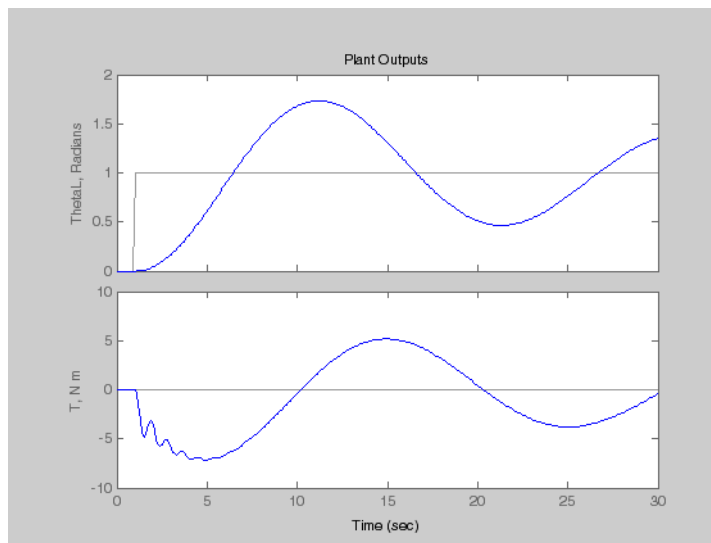
Response to Unit Step in the Angular Position Setpoint
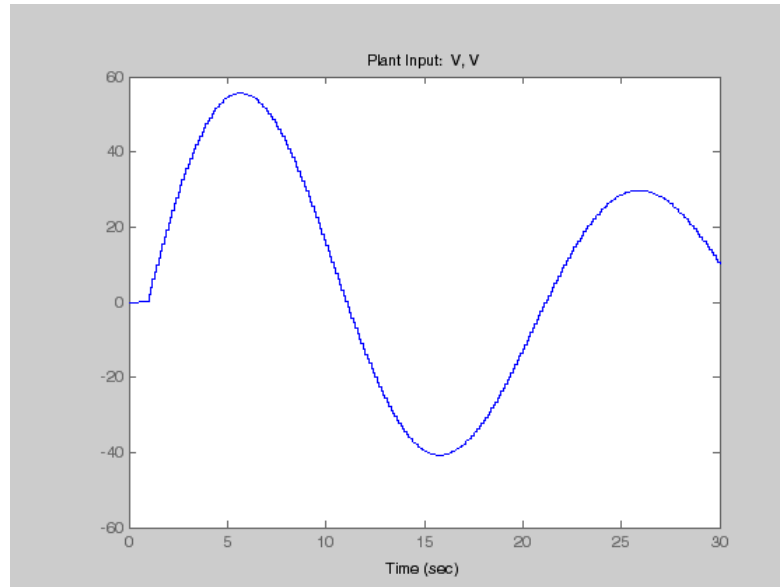
Note The window shown in Response to Unit Step in the Angular Position Setpoint on page 4-14 provides many of the customization features available in Control System Toolbox `ltiview` and `sisotool` displays. Try clicking a curve to obtain the numerical characteristics of the selected point, or right-clicking in the plot area to open a customization menu.

The corresponding applied voltage adjustments appear in a separate window and are also very sluggish.

On the positive side, the applied torque stays well within bounds, as does the applied voltage.

Retuning to Achieve a Faster Servo Response

To obtain a more rapid servo response, navigate to the **MPC1 Weight Tuning** pane (select the **MPC1** node to get the controller design view, then click the **Weight Tuning** tab) and move the slider all the way to the right. Then click the green arrow in the toolbar. Your results should now resemble Faster Servo Response on page 4-16 and Manipulated Variable Adjustments on page 4-17.

The angular position now settles within 10 seconds following the step. The torque approaches its lower limit, but doesn't exceed it (see Faster Servo Response on page 4-16) and the applied voltage stays within its limits (see Manipulated Variable Adjustments on page 4-17).
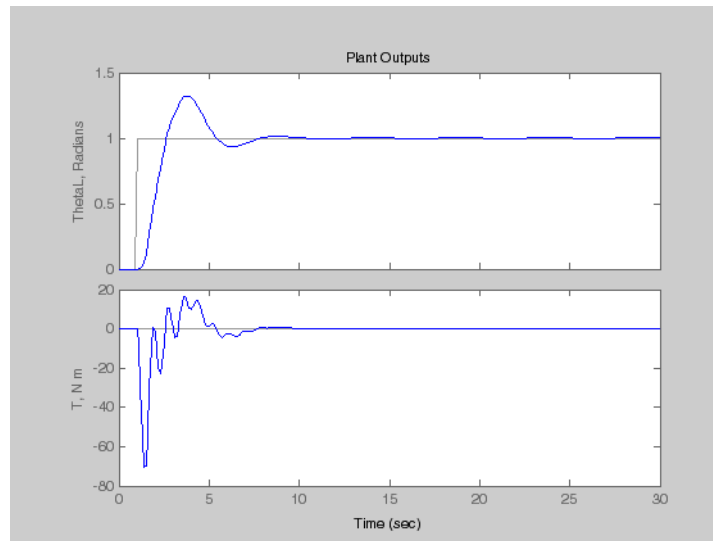
Faster Servo Response
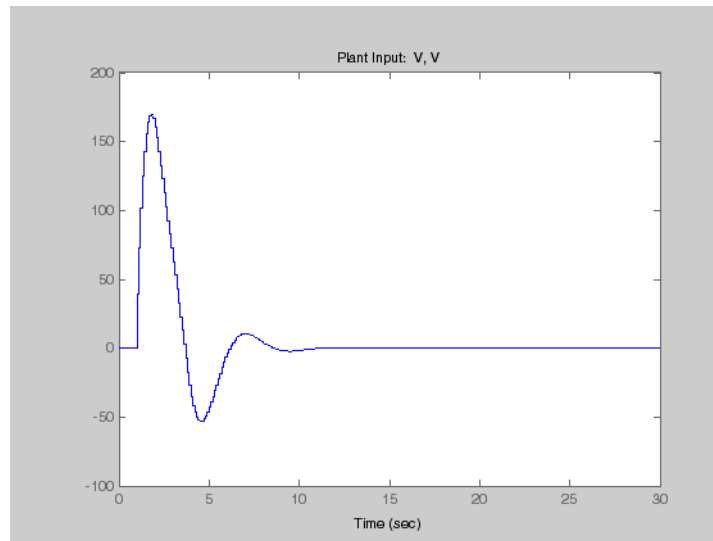
Manipulated Variable Adjustments

Modifying the Scenario

Finally, increase the step size to π radians (select the **Scenario1** node and edit the tabular value).

As shown in Servo Response for Step Increase of π Radians on page 4-18 and Voltage Adjustments on page 4-18, the servo response is essentially as good as before, and we avoid exceeding the torque constraint at –78.5 Nm, even though the applied voltage is saturated for about 2.5 seconds (see Voltage Adjustments on page 4-18).
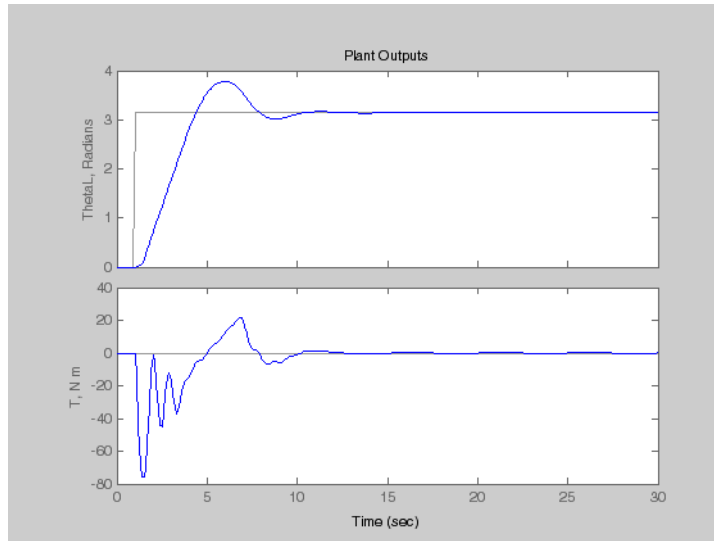
Servo Response for Step Increase of   Radians
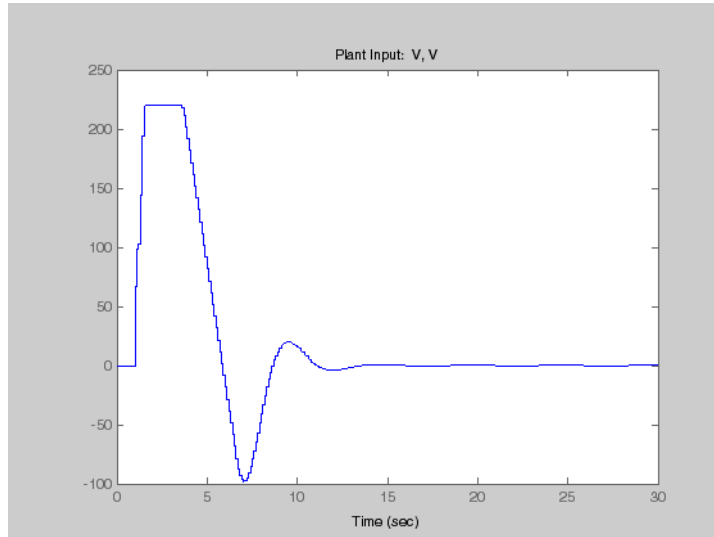
Voltage Adjustments

Saving Your Work

Once you're satisfied with a controller's performance, you can export it to the workspace, for use in a Simulink block diagram or for analysis (or you can save it in a MAT-file).

To export a controller, right-click its node and select **Export** from the resulting menu (or select the **Controllers** node, select the controller in the list, and click the **Export** button). A dialog box like that shown in Exporting a Controller to the Workspace on page 4-19 will appear.

The **Controller source** is the design from which you want to extract a controller. There's only one in this example, but in general you might be working on several simultaneously. The **Controller to export** choice defaults to the controller most recently selected. Again, there's no choice in this case, but there could be in general. The **Name to assign** edit box allows you to rename the exported controller. (This will not change its name in the design tool.)

Exporting a Controller to the Workspace

Note When you exit the design tool, you will be prompted to save the entire design in a MAT file. This allows you to reload it later using the **File/Load** menu option or the **Load** icon on the toolbar.

Using Model Predictive Control Toolbox Commands

Once you've become familiar with the toolbox, you may find it more convenient to build a controller and run a simulation using commands.

For example, suppose that you've defined the model as discussed in "Defining the Plant Model" on page 4-4. Consider the following command sequence:

```
ManipulatedVariables = struct('Min', -220, 'Max', 220, 'Units', 'V');
OutputVariables(1) = struct('Min', -Inf, 'Max', Inf, 'Units', 'rad');
OutputVariables(2) = struct('Min', -78.5, 'Max', 78.5, 'Units', 'Nm');
Weights = struct('Input', 0, 'InputRate', 0.05, 'Output', [10 0]);
Model.Plant = sys;
Model.Plant.OutputGroup = {[1], 'Measured' ; [2], 'Unmeasured'};
Ts = 0.1;
PredictionHorizon = 10;
ControlHorizon = 2;
```

This creates several *structure* variables. For example, ManipulatedVariables defines the display units and constraints for the applied voltage (the manipulated plant input). Weights defines the tuning weights shown in Controller Design View, Weight Tuning Pane on page 4-12 (but the numerical values used here provide better performance). Model designates the plant model (stored in sys, which we defined earlier). The code also sets the Model.Plant.OutputGroup property to designate the second output as unmeasured.

Constructing an MPC Object

Use the mpc command to construct an MPC object called ServoMPC:

```
ServoMPC = mpc(Model, Ts, PredictionHorizon, ControlHorizon);
```

Like the LTI objects used to define linear, time-invariant dynamic models, an MPC object contains a complete definition of a controller.

Setting, Getting, and Displaying Object Properties

Once you've constructed an MPC object, you can change its properties as you would for other objects. For example, to change the prediction horizon, you could use one of the following commands:

```
ServoMPC.PredictionHorizon = 12;
set(ServoMPC, 'PredictionHorizon', 12);
```

For a listing of all the object's properties, you could type:

```
get(ServoMPC)
```

To access a particular property (e.g., the control horizon), you could type either:

```
M = get(ServoMPC, 'ControlHorizon');
M = ServoMPC.ControlHorizon;
```

You can also set multiple properties simultaneously.

Set the following properties before continuing with this example:

```
set(ServoMPC, 'Weights', Weights, ...
 'ManipulatedVariables', ManipulatedVariables, ...
 'OutputVariables', OutputVariables);
```

Typing the name of an object without a terminating semicolon generates a formatted display of the object's properties. You can achieve the same effect using the display command:

```
display(ServoMPC)
```

Running a Simulation

The sim command performs a linear simulation. For example, the following code sequence defines constant setpoints for the two outputs, then runs a simulation:

```
TimeSteps = round(10/Ts);
r = [pi 0];
[y, t, u] = sim(ServoMPC, TimeSteps, r);
```

By default, the model used to design the controller (stored in ServoMPC) also represents the plant.

The sim command saves the output and manipulated variable sequences in variables y and u. For example,

```
subplot(311)
plot(t, y(:,1), [0 t(end)], pi*[1 1])
```

```
title('Angular Position (radians)');
subplot(312)
plot(t, y(:,2), [0 t(end)], [-78.5 -78.5])
title('Torque (nM)')
subplot(313)
stairs(t, u)
title('Applied Voltage (volts)')
xlabel('Elapsed Time (seconds)')
```

produces the custom plot shown in Plotting the Output of the sim Command on page 4-22. The plot includes the angular position's setpoint. The servo response settles within 5 seconds with no overshoot. It also displays the torque's lower bound, which becomes active after about 0.9 seconds but isn't exceeded. The applied voltage saturates between about 0.5 and 2.8 seconds, but the controller performs well despite this.
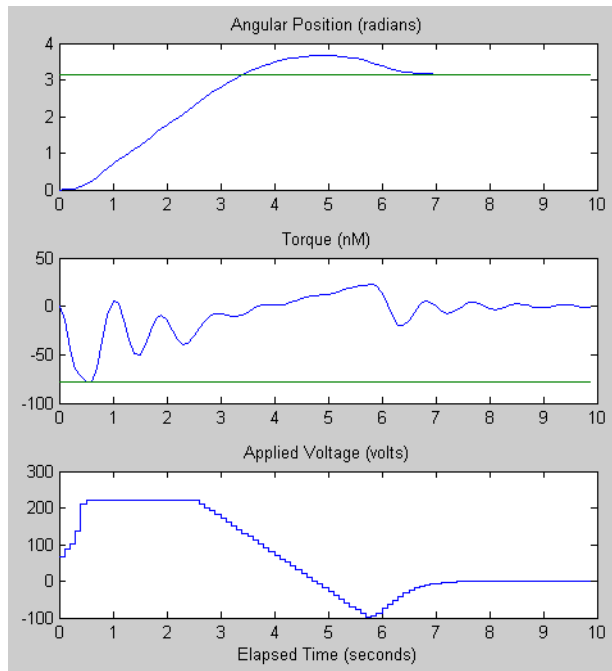
Plotting the Output of the sim Command

Using MPC Tools in Simulink

Block Diagram for the Servomechanism Example on page 4-23 is a Simulink block diagram for the servomechanism example. Most of the blocks are from the standard Simulink library. There are two exceptions:

- Servomechanism Model is an LTI System block from the Control System Toolbox library. The LTI model sys (which must exist in the workspace) defines its dynamic behavior. To review how to create this model, see "Defining the Plant Model" on page 4-4.

- MPC Controller is from the MPC Blocks library. Model Predictive Control Toolbox™ Simulink® Block Dialog Box on page 4-24 shows the dialog box obtained by double-clicking this block. You need to supply an MPC object, and ServoMPC is being used here. It must be in the workspace before you run a simulation. The **Design** button opens the design tool, which allows you to create or modify the object. To review how to use commands to create ServoMPC, see "Constructing an MPC Object" on page 4-20.
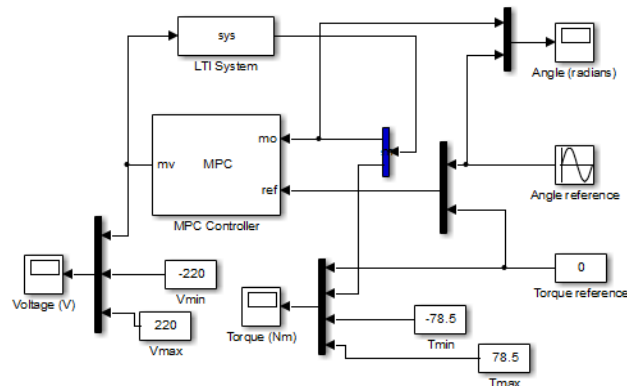
Block Diagram for the Servomechanism Example
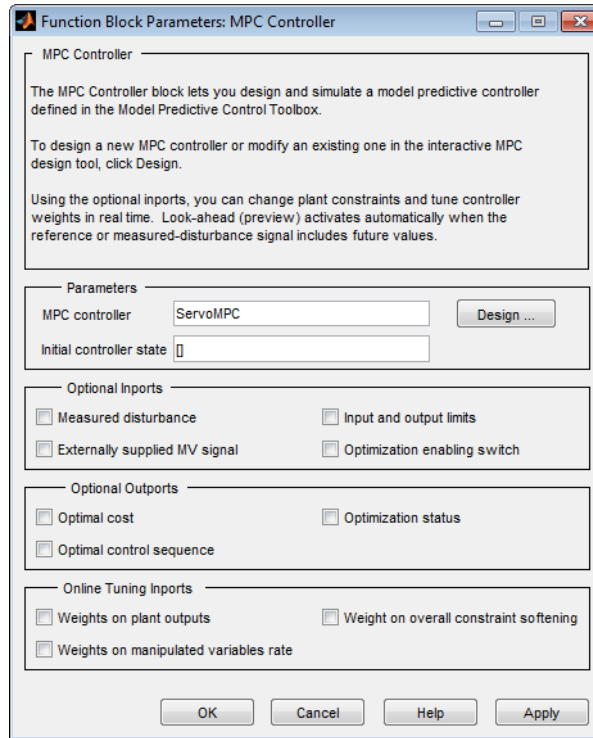
Model Predictive Control Toolbox™ Simulink® Block Dialog Box

The key features of the diagram are as follows:

- The MPC Controller output is the plant input. The Voltage Scope block plots it (yellow curve). Minimum and maximum voltage values are shown as magenta and cyan curves.

- The plant output is a vector signal. The first element is the measured angular position. The second is the unmeasured torque. A Demux block separates them. The angular position feeds back to the controller and plots on the Angle scope (yellow curve). The torque plots on the Torque scope (with its lower and upper bounds).

- The position setpoint varies sinusoidally with amplitude π radians and frequency 0.4 rad/s. It also appears on the Angle scope (magenta curve).
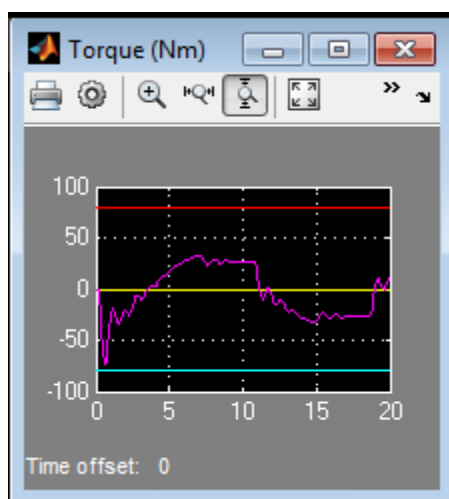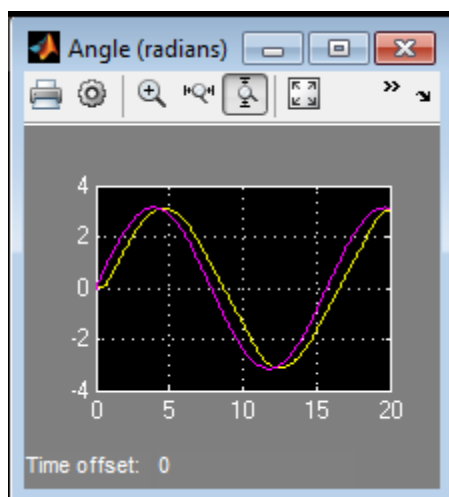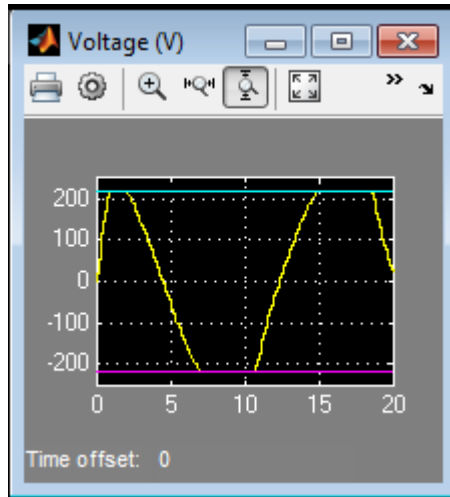
The angular position tracks the sinusoidal setpoint variations well despite saturation of the applied voltage. The setpoint variations are more gradual than the step changes used previously, so the torque stays well within its bounds.

Paper Machine Process Control

System Model

Ying *et al.* [1] studied the control of consistency (percentage pulp fibers in aqueous suspension) and liquid level in a paper machine headbox, a schematic of which is shown in Schematic of Paper Machine Headbox Elements on page 4-27.
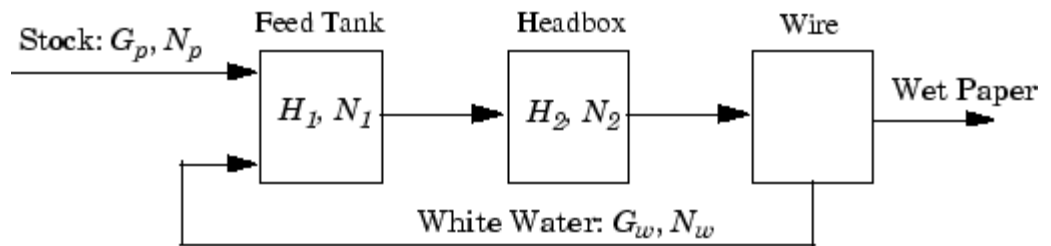
Schematic of Paper Machine Headbox Elements

The process is nonlinear, and has three outputs, two manipulated inputs, and two disturbance inputs, one of which is measured for feedforward control.

The process model is a set of ordinary differential equations (ODEs) in bilinear form. The states are

$$x = \begin{bmatrix} H_1 & H_2 & N_1 & N_2 \end{bmatrix}^T$$

where H_1 is the liquid level in the feed tank, H_2 is the headbox liquid level, N_1 is the feed tank consistency, and N_2 is the headbox consistency. The measured outputs are:

$$y = \begin{bmatrix} H_2 & N_1 & N_2 \end{bmatrix}^T$$

The primary control objectives are to hold H_2 and N_2 at setpoints. There are two manipulated variables

$$u = \begin{bmatrix} G_p & G_w \end{bmatrix}^T$$

where G_p is the flow rate of stock entering the feed tank, and G_w is the recycled *white water* flow rate. The consistency of stock entering the feed tank, N_p, is a measured disturbance.

$v = N_p$

The white water consistency is an unmeasured disturbance.

$d = N_w$

Variables are normalized. All are zero at the nominal steady state and have comparable numerical ranges. Time units are minutes. The process is open-loop stable.

The mpcdemos folder contains the file mpc_pmmodel.m, which implements the nonlinear model equations as a Simulink S-function. The input sequence is G_p, G_w, N_p, N_w, and the output sequence is $_2$, N_1, N_2.

Linearizing the Nonlinear Model

The paper machine headbox model is easy to linearize analytically, yielding the following state space matrices:

```
A = [-1.9300          0          0          0
      0.3940    -0.4260          0          0
          0          0    -0.6300          0
      0.8200    -0.7840     0.4130    -0.4260];
B = [1.2740     1.2740          0          0
          0          0          0          0
      1.3400    -0.6500     0.2030     0.4060
          0          0          0          0];
```

```
C = [0    1.0000         0         0
          0         0    1.0000         0
          0         0         0    1.0000];
D = zeros(3,4);
```

Use these to create a continuous-time LTI state-space model, as follows:

```
PaperMach = ss(A, B, C, D);
PaperMach.InputName = {'G_p', 'G_w', 'N_p', 'N_w'};
PaperMach.OutputName = {'H_2', 'N_1', 'N_2'};
```

(The last two commands are optional; they improve plot labeling.)

As a quick check of model validity, plot its step responses as follows:

```
step(PaperMach);
```

The results appear in the following figure. Note the following:

- The two manipulated variables affect all three outputs.
- They have nearly identical effects on H_2.
- The $G_w \rightarrow N_2$ pairing exhibits an inverse response.

These features make it difficult to achieve accurate, independent control of H_2 and N_2.
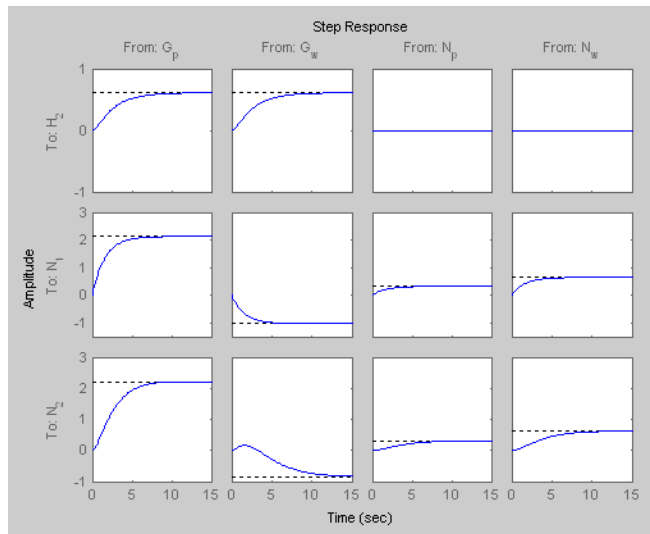
Linearized Paper Machine Model's Step Responses

MPC Design

Type

```
mpctool
```

to open the MPC design tool. Import your LTI Paper Mach model as described in "Opening MPCTOOL and Importing a Model" on page 4-6.

Next, define signal properties, being sure to designate N_p and N_w as measured and unmeasured disturbances, respectively. Your specifications should resemble Signal Properties for the Paper Machine Application on page 4-31.

Input signal properties				
Name	Type	Description	Units	Nominal
G_p	Manipulated	Feed flow rate	kg/h	0.0
G_w	Manipulated	White water flow rate	kg/h	0.0
N_p	Meas. disturb.	Feed consistency	%	0.0
N_w	Unmeas. disturb.	White water consistency	%	0.0

Output signal properties				
Name	Type	Description	Units	Nominal
H_2	Measured	Headbox level	m	0.0
N_1	Measured	Feed tank consistency	%	0.0
N_2	Measured	Head box consistency	%	0.0

Signal Properties for the Paper Machine Application

Initial Controller Design

If necessary, review "Specifying Controller Properties" on page 4-9. Then click the **MPC1** node and specify the following controller parameters (leaving others at their default values):

- **Models and Horizons**. Control interval = 2 minutes

- **Constraints**. For both G_p and G_w, Minimum = –10, Maximum = 10, Max down rate = –2, Max up rate = 2.

- **Weight Tuning**. For both G_p and G_w, Weight = 0, Rate weight = 0.4. For N_1, Weight = 0. (Other outputs have Weight = 1.)

Servo Response

Finally, select the **Scenario1** node and define a servo-response test:

- Duration = 30

- H_2 setpoint = 1 (constant)

Simulate the scenario. You should obtain results like those shown in Servo Response for Unit Step in Headbox Level Setpoint on page 4-32 and Manipulated Variable Moves on page 4-32.
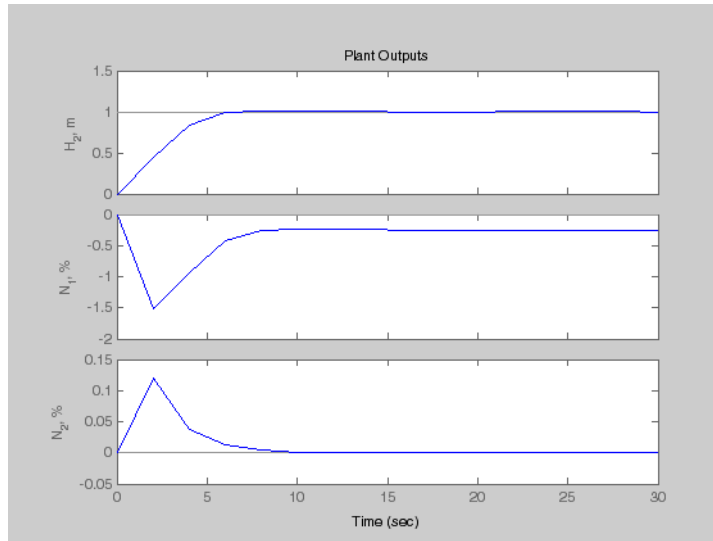
Servo Response for Unit Step in Headbox Level Setpoint
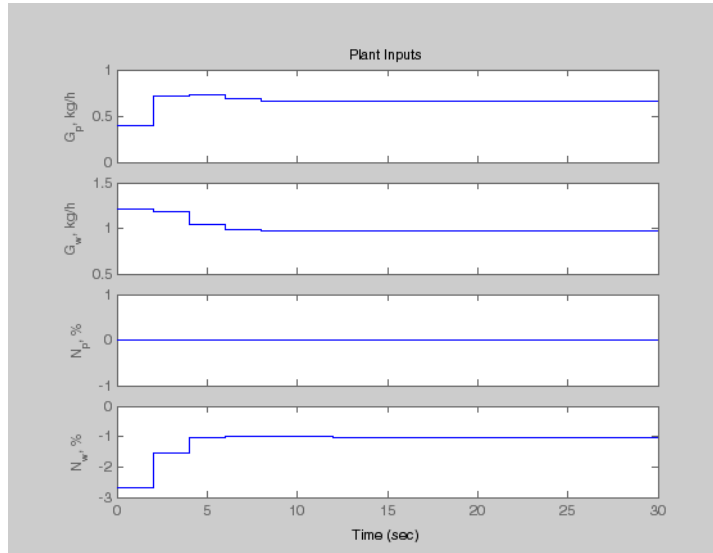
Manipulated Variable Moves

Weight Tuning

The response time is about 8 minutes. We could reduce this by decreasing the control interval, reducing the manipulated variable rate weights, and/or eliminating the up/down rate constraints. The present design uses a conservative control effort, which would usually improve robustness, so we will continue with the current settings.

Note the steady-state error in N_1 (it's about –0.25 units in Servo Response for Unit Step in Headbox Level Setpoint on page 4-32). There are only two manipulated variables, so it's impossible to hold three outputs at setpoints. We don't have a setpoint for N_1 so we have set its weight to zero (see controller settings in "Initial Controller Design" on page 4-31). Otherwise, all three outputs would have exhibited steady-state error (try it).

Consistency control is more important than level control. Try decreasing the H_2 weight from 1 to 0.2. You should find that the peak error in N_2 decreases by almost an order of magnitude, but the H_2 response time increases from 8 to about 18 minutes (not shown). Use these modified output weights in subsequent tests.

Feedforward Control

To configure a test of the controller's feedforward response, define a new scenario by clicking the **Scenarios** node, clicking the **New** button, and renaming the new scenario **Feedforward** (by editing its name in the tree or the summary list).

In the **Feedforward** scenario, define a step change in the measured disturbance, N_p, with **Initial value** = 0, **Size** = 1, **Time** = 10. All output setpoints should be zero. Set the **Duration** to 30 time units.

If response plots from the above servo response tests are still open, close them. Simulate the **Feedforward** scenario. You should find that the H_2 and N_2 outputs deviate very little from their setpoints (not shown).

Experiment with the "look ahead" feature. First, observe that in the simulation just completed the manipulated variables didn't begin to move until the disturbance occurred at t = 10 minutes. Return to the **Feedforward** scenario, select the **Look ahead** option for the measured disturbance, and repeat the simulation.

Notice that the manipulated variables begin changing *in advance* of the disturbance. This happens because the look ahead option uses known future values of the disturbance when computing its control action. For example, at time $t = 0$ the controller is using a prediction horizon of 10 control intervals (20 time units), so it "sees" the impending disturbance at $t = 10$ and begins to prepare for it. The output setpoint tracking improves slightly, but it was already so good that the improvement is insignificant. Also, it's unlikely that there would be advanced knowledge of a consistency disturbance, so clear the **Look ahead** check box for subsequent simulations.

Unmeasured Input Disturbance

To test the response to unmeasured disturbances, define another new scenario called **Feedback**. Configure it as for **Feedforward**, but set the measured disturbance, N_p, to zero (constant), and the unmeasured disturbance, N_w, to 1.0 (constant). This simulates a sudden, sustained, unmeasured disturbance occurring at time zero.

Running the simulation should yield results like those in Feedback Scenario: Unmeasured Disturbance Rejection on page 4-35. The two controlled outputs (H_2 and N_2) exhibit relatively small deviations from their setpoints (which are zero). The settling time is longer than for the servo response (compare to Servo Response for Unit Step in Headbox Level Setpoint on page 4-32) which is typical.
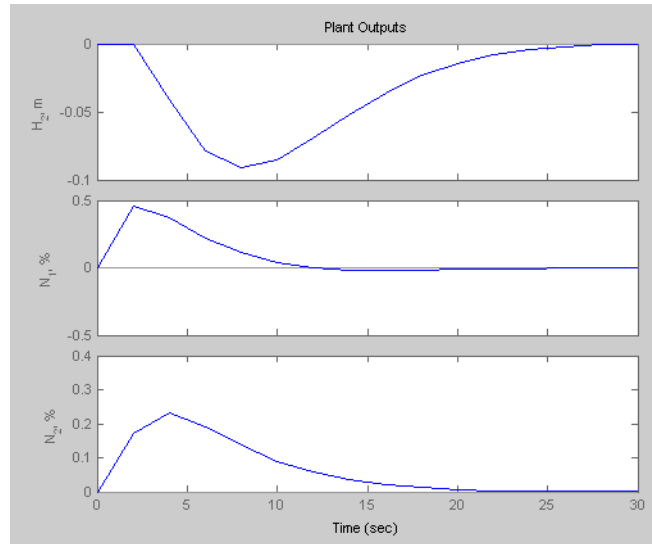
Feedback Scenario: Unmeasured Disturbance Rejection

One factor limiting performance is the chosen control interval of 2 time units. The controller can't respond to the disturbance until it first appears in the outputs, i.e., at $t = 2$. If you wish, experiment with larger and smaller intervals (modify the specification on the controller's **Model and Horizons** tab).

Effect of Estimator Assumptions

Another factor influencing the response to unmeasured disturbances (and model prediction error) is the estimator configuration. The results shown in Feedback Scenario: Unmeasured Disturbance Rejection on page 4-35 are for the default configuration.

To view the default assumptions, select the controller node (**MPC1**), and click its **Estimation** tab. The resulting view should be as shown in Default Estimator Assumptions: Output Disturbances on page 4-36. The status message (bottom of figure) indicates that Model Predictive Control Toolbox default assumptions are being used.

Name	Units	Type	Magnitude
H_2	m	Steps	1.0
N_1	%	Steps	1.0
N_2	%	White	0.0
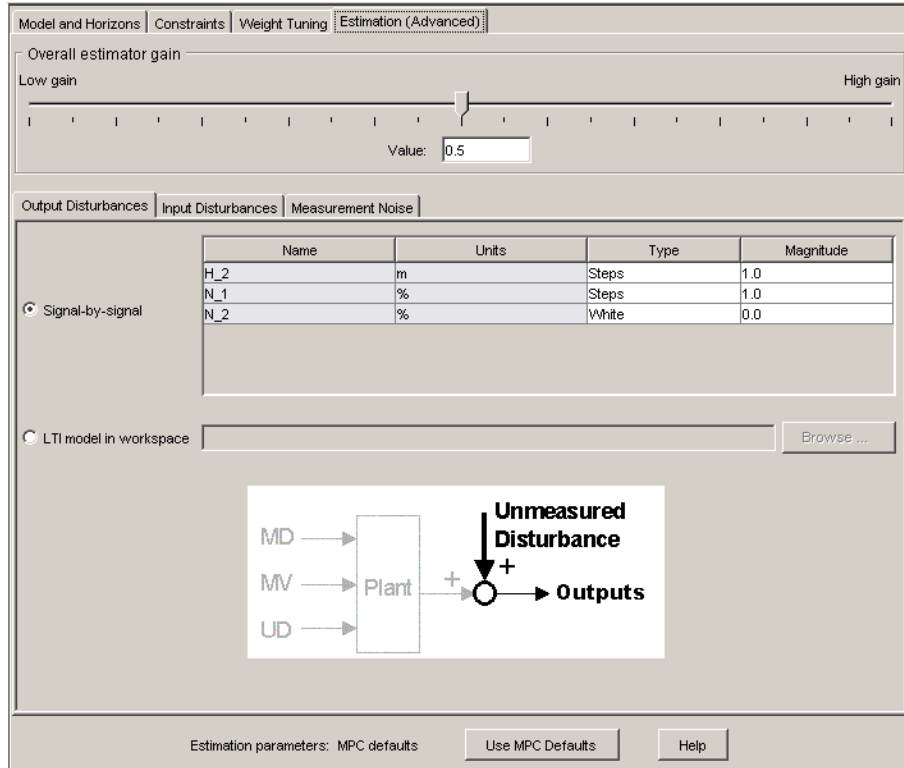
Default Estimator Assumptions: Output Disturbances

Now consider the upper part of the figure. The **Output Disturbances** tab is active, and its **Signal-by-signal** option is selected. According to the tabular data, the controller is assuming independent, step-like disturbances (i.e., integrated white noise) in the first two outputs.

Click the **Input Disturbances** tab. Verify that the controller is also assuming independent step-like disturbances in the unmeasured disturbance input.

Thus, there are a total of three independent, sustained (step-like) disturbances. This allows the controller to eliminate offset in all three measured outputs.

The disturbance magnitudes are unity by default. Making one larger than the rest would signify a more important disturbance at that location.

Click the **Measurement Noise** tab. Verify that white noise (unit magnitude) is being added to each output. The noise magnitude governs how much influence each measurement has on the controller's decisions. For example, if a particular measurement is relatively noisy, the controller will give it less weight, relying instead upon the model predictions of that output. This provides a noise filtering capability.

In the paper machine application, the default disturbance assumptions are reasonable. It is difficult to improve disturbance rejection significantly by modifying them.

Controlling the Nonlinear Plant in Simulink

It's good practice to run initial tests using the linear plant model as described in "Servo Response" on page 4-31 and "Unmeasured Input Disturbance" on page 4-34. Such tests don't introduce prediction error, and are a useful benchmark for more demanding tests with a nonlinear plant model. The controller's prediction model is linear, so such tests introduce prediction error.

Open the paper machine headbox control Simulink model by typing:

mpc_papermachine
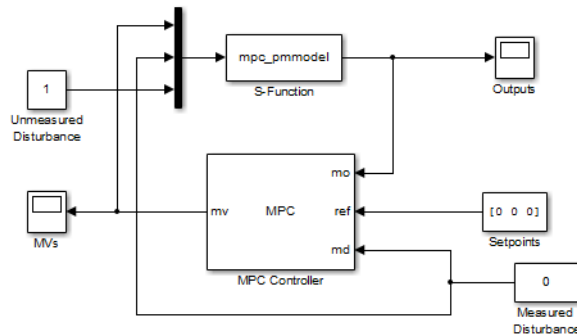
Paper Machine Headbox Control Using MPC Tools in Simulink®

Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-37 is a Simulink model in which the Model Predictive Control Toolbox controller is being used to regulate the nonlinear paper machine headbox model. The block labeled S-Function embodies the nonlinear model, which is coded in a file called `mpc_pmmodel.m`.

As shown in the following dialog box, the MPC block references a controller design called MPC1, which was exported to the MATLAB workspace from the design tool. Note also that the measured disturbance inport is enabled, allowing the measured disturbance to be connected as shown in Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-37.
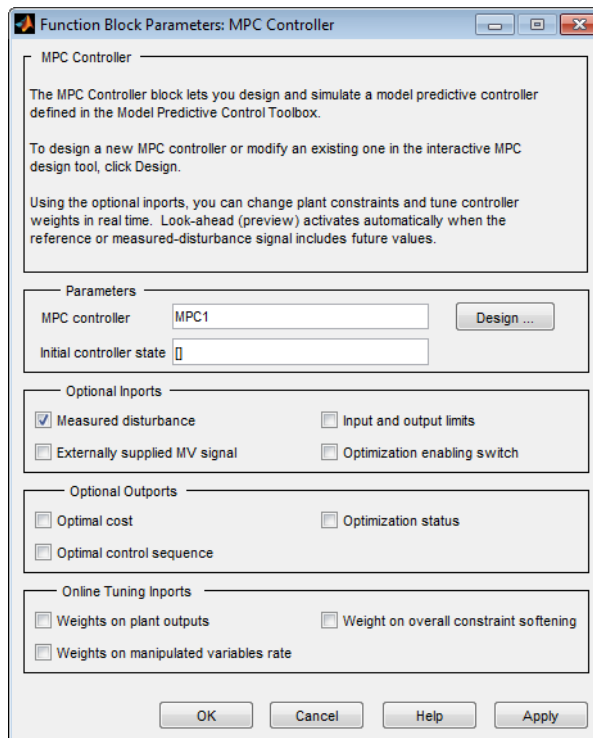
Test, Output Variables on page 4-39 shows the scope display from the "Outputs" block for the setup of Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-37, i.e., an unmeasured disturbance. The yellow curve is H_2, the magenta is N_1, and the cyan is N_2. Comparing to Feedback

Scenario: Unmeasured Disturbance Rejection on page 4-35, the results are almost identical, indicating that the effects of nonlinearity and prediction error were insignificant in this case. Simulink® Test, Manipulated Variables on page 4-40 shows the corresponding manipulated variable moves (from the "MVs" scope in Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-37) which are smooth yet reasonably fast.

As disturbance size increases, nonlinear effects begin to appear. For a disturbance size of 4, the results are still essentially the same as shown in Test, Output Variables on page 4-39 and Simulink® Test, Manipulated Variables on page 4-40 (scaled by a factor of 4), but for a disturbance size of 6, the setpoint deviations are relatively larger, and the curve shapes differ (not shown). There are marked qualitative and quantitative differences when the disturbance size is 8. When it is 9, deviations become very large, and the MVs saturate. If such disturbances were likely, the controller would have to be retuned to accommodate them.
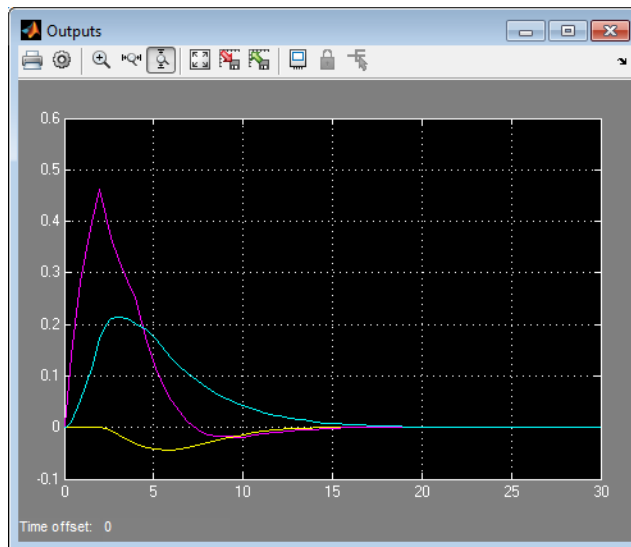
Test, Output Variables
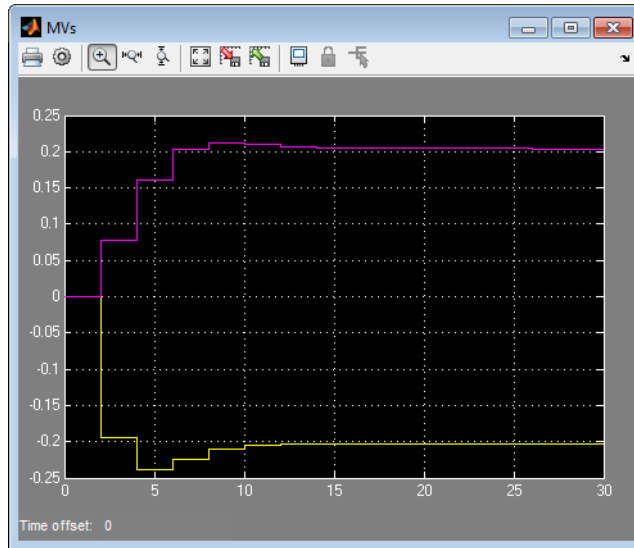
Simulink® Test, Manipulated Variables

References

[1] Ying, Y., M. Rao, and Y. Sun "Bilinear control strategy for paper making process," *Chemical Engineering Communications* (1992), Vol. 111, pp. 13–28.

Transfer Bumplessly Between Manual and Automatic Operations

This example shows a bumpless transfer between manual and automatic operation. The context is a Simulink simulation involving the MPC Controller block and a single-input, single-output, LTI plant.

During startup of a continuous plant, the operators often adjust key actuators manually until the plant is near the desired operating point, and then switch to automatic control. If not done correctly, the transfer can cause a *bump*, i.e., large actuator movements.

A Model Predictive Controller must monitor all known plant signals even when it is not in control of the actuators. This improves its state estimates and allows a bumpless transfer to automatic operation.

Open the Simulink model.

```
open_system('mpc_bumpless');
```
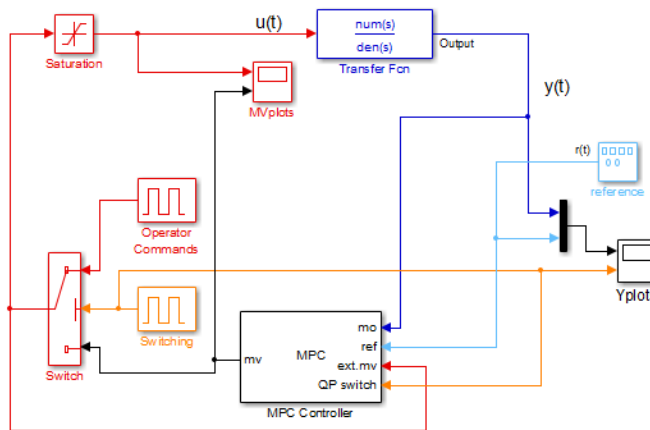

Simulink® Block Diagram for the MPC Bumpless Transfer Example

Create the plant model.

```
num=[1 1];
den=[1 3 2 0.5];
```

```
sys=tf(num,den);
```

The plant is a stable single-input single-output system as seen in its step response.
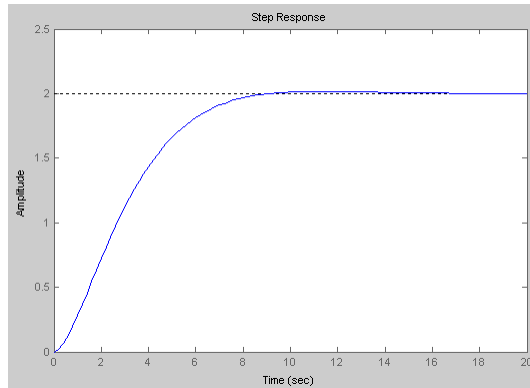
Create an MPC controller.

```
Ts=0.5;      % sampling time
p=15;        % prediction horizon
m=2;         % control horizon
MPC1=mpc(sys,Ts,p,m);
MPC1.Weights.Output=0.01;
MPC1.MV=struct('Min',-1,'Max',1,'RateMin',-1e5);
Tstop=250;
```

The MPC controller object, MPC1, uses a sampling period of 0.5 seconds. In the dialog for the MPC Controller block, specify MPC1 in the **MPC controller** box.

As shown in MPC Block Configuration Settings on page 4-43, the block's optional input port for externally supplied manipulated variables is selected. This adds the inport labeled ext.mv to the block (Simulink® Block Diagram for the MPC Bumpless Transfer Example on page 4-41 shows how this is connnected).

The optional input port for switching off the optimization is also selected, which adds the inport labeled QP switch to the block (see Simulink® Block Diagram for the MPC Bumpless Transfer Example on page 4-41).
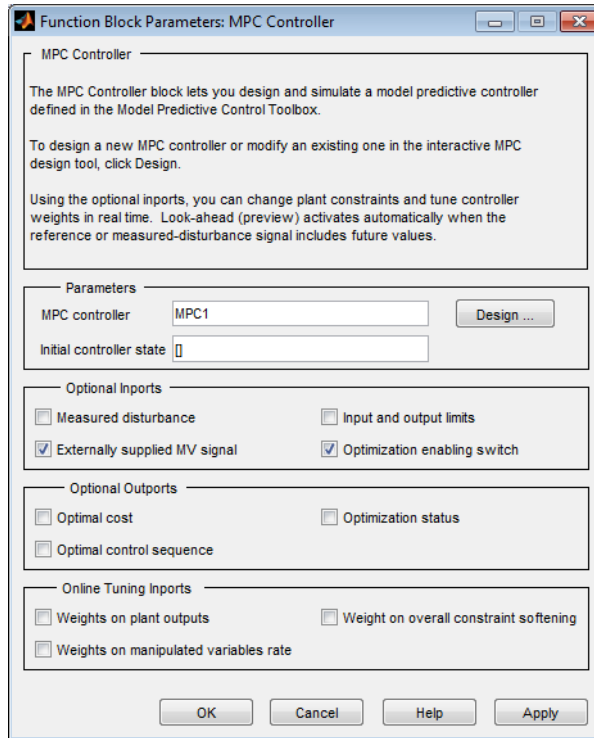
MPC Block Configuration Settings

The example tests the effect of switching the controller from automatic to manual and back. To simulate this, a Pulse Generator block labeled `switching signal` sends either one or zero to a switch. When it sends zero, the system is in automatic mode, and the MPC block's output goes to the plant. Otherwise, the system is in manual mode, and the signal from the Operator Commands block goes to the plant.

In both cases the actual plant input feeds back to the controller, as shown in Simulink® Block Diagram for the MPC Bumpless Transfer Example on page 4-41 (unless the plant input saturates at –1 or 1). The controller also monitors the plant ouput at all times. Thus, the controller can update its estimate of the plant state even when in manual.

The example also employs the optimization switching option. When the system switches to manual, a nonzero signal enters the controller's QP

Switch inport, turning off the optimization calculations, thereby reducing computational effort. The benefit is small in this trivial example but it could be significant in a demanding real-time application.

As shown in Output, Reference and Switching Signal on page 4-44, the system is in automatic mode for the first 90 time units (switching signal is zero). During this time the controller smoothly drives the controlled plant output from its initial value, 0, to the desired reference value, –0.5.
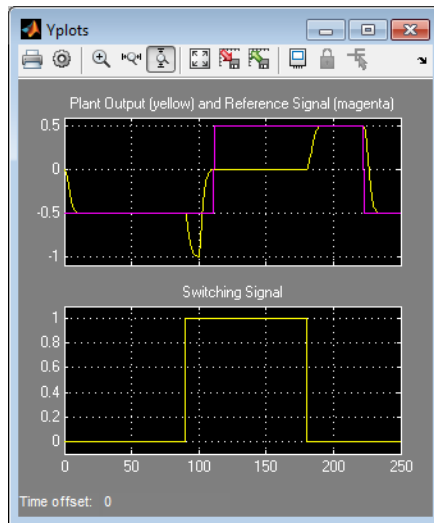
Output, Reference and Switching Signal

At time 90, manual operation begins (switching signal goes from zero to one). This causes the Switch element to send the operator commands to the plant instead of the controller output.

Simultaneously, the nonzero signal entering the controller's `QP Switch` inport turns off the optimization calculations and the controller output goes to zero (see Manipulated Variable (Actuator) Adjustments on page 4-45).

Once in manual mode, the operator commands set the manipulated variable to –0.5 for 10 time units, and then to 0. Output, Reference and Switching Signal on page 4-44 shows the open-loop response between times 90 and 180 when the controller is deactivated.
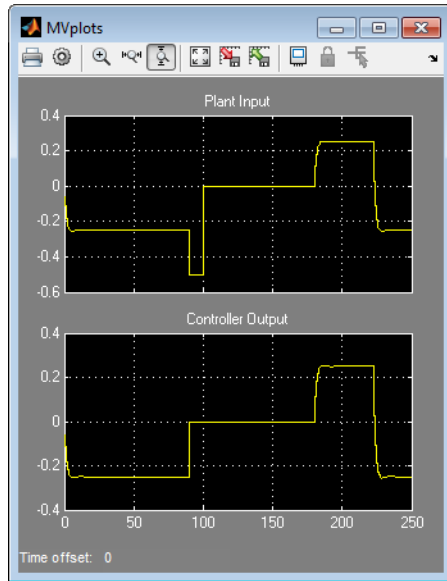
Manipulated Variable (Actuator) Adjustments

At time 180, the system switches back to automatic mode. Output, Reference and Switching Signal on page 4-44 shows that the output returns to the reference value smoothly, and Manipulated Variable (Actuator) Adjustments on page 4-45 shows similarly smooth adjustments in the controller output.

Note that the controller's state estimator has default zero initial conditions, which are appropriate when this simulation begins. Thus, there is no bump at startup. In general you should start the system running in manual mode for long enough to allow the controller to acquire an accurate state estimate before switching to automatic mode.

Now consider the situation shown in External Manipulated Variable Feedback Turned Off on page 4-46. The external manipulated variable feedback has been turned off. This causes the controller to assume that its adjustments are always going to the plant, which is incorrect whenever the system switches to manual mode.

The controller's `QP Switch` is also turned off.
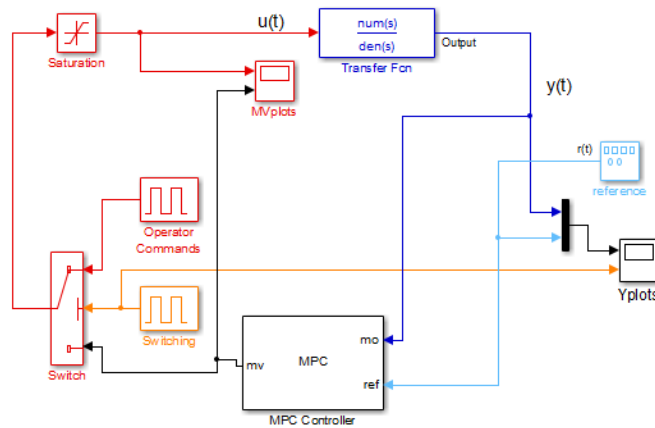
External Manipulated Variable Feedback Turned Off

As shown in Output Response with Manipulated Variable Feedback and QP Switching Turned Off on page 4-47 and Manipulated Variable Adjustments with Manipulated Variable Feedback Disconnected and QP Switching Manipulated Variable Feedback and QP Switching Turned Off on page 4-48, the behavior is identical to the original case for the first 90 time units (compare to Output, Reference and Switching Signal on page 4-44 and Manipulated Variable (Actuator) Adjustments on page 4-45).

When the system switches to manual between 90 and 180, the plant behavior is the same as before but the controller tries in vain to hold the plant at the setpoint. Consequently, its output increases and eventually saturates. It assumes this output is going to the plant, so its state estimates become inaccurate. Thus, when the system switches to automatic mode at time 180, there is a large bump.

The benefits of bumpless transfer are evident.
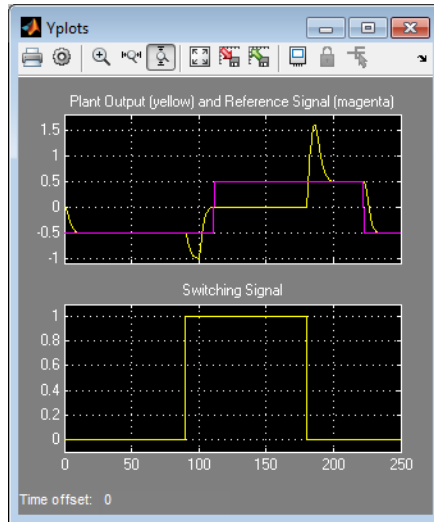
Output Response with Manipulated Variable Feedback and QP Switching Turned Off
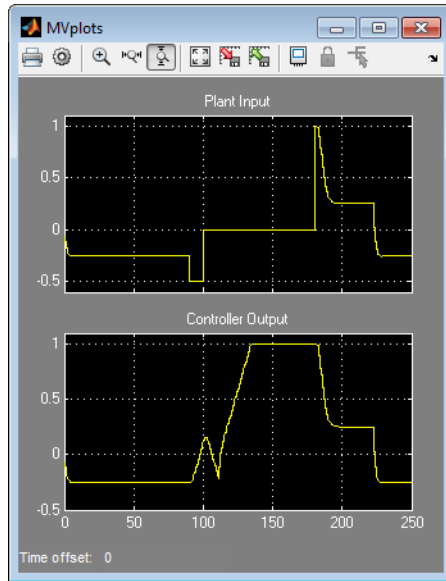
Manipulated Variable Adjustments with Manipulated Variable Feedback Disconnected and QP Switching Manipulated Variable Feedback and QP Switching Turned Off

Coordinate Multiple Controllers at Different Operating Points

Chemical reactors can exhibit strongly nonlinear behavior due to the exponential effect of temperature on reaction rate. If the primary reaction is exothermic, an increase in reaction rate causes an increase in reactor temperature. This positive feedback can lead to open-loop unstable behavior.

Reactors operate in either a continuous or a batch mode. In batch mode, operating conditions can change dramatically during a batch as the reactants disappear. Although continuous reactors typically operate at steady state, they must often move to a new steady state. In other words, both batch and continuous reactors need to operate safely and efficiently over a range of conditions.

If the reactor behaves nonlinearly, a single linear controller might not be able to manage such transitions. One approach is to develop linear models that cover the anticipated operating range, design a controller based on each model, and then define a criterion by which the control system switches from one such controller to another. Gain scheduling is an established technique. The challenge is to move the reactor operating conditions from an initial steady-state point to a much different condition. The transition passes through a region in which the plant is open-loop unstable. This example illustrates an alternative — coordination of multiple MPC controllers. The solution uses the Simulink Multiple MPC Controller block to coordinate the use of three controllers, each of which has been designed for a particular operating region.

The subject process is a constant-volume continuous stirred-tank reactor (CSTR). The model consists of two nonlinear ordinary differential equations (see [1]). The model states are the reactor temperature and the rate-limiting reactant concentration. For the purposes of this example, both are assumed to be measured plant outputs.

There are three inputs:

• Concentration of the limiting reactant in the reactor feed stream, $kmol/m^3$

• The reactor feed temperature, K

- The coolant temperature, K

The control system can adjust the coolant temperature in order to regulate the reactor state and the rate of the exothermic main reaction. The other two inputs are independent unmeasured disturbances.

The Simulink diagram for this example appears below. The CSTR model is a masked subsystem. The feed temperature and composition are constants. As discussed above, the control system adjusts the coolant temperature (the `Tc` input on the CSTR block).
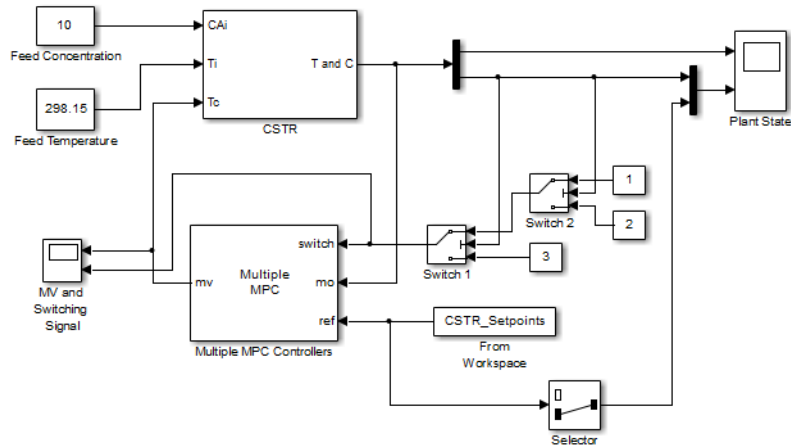
The two CSTR outputs are the reactor temperature and composition respectively. These are being sent to a scope display and to the control system as feedback.

The reference signal (i.e. setpoint) is coming from variable CSTR_Setpoints, which is in the base workspace. As there is only one manipulated variable (the coolant temperature) the control objective is to force the *reactor concentration* to track a specified trajectory. The concentration setpoint also goes to the `Plant State` scope for plotting. The control system receives a setpoint for the reactor temperature too but the controller design ignores it.

In that case why supply the temperature measurement to the controller? The main reason is to improve state estimation. If this were not done, the control

system would have to infer the temperature value from the concentration measurement, which would introduce an estimation error and degrade the model's predictive accuracy.

The rationale for the `Switch 1` and `Switch 2` blocks appears below.

The figure below shows the `Multi MPC Controller` mask. The block is coordinating three controllers (`MPC1`, `MPC2` and `MPC3` in that sequence). It is also receiving the setpoint signal from the workspace, and the **Look ahead** option is active. This allows the controller to anticipate future setpoint values and usually improves setpoint tracking.
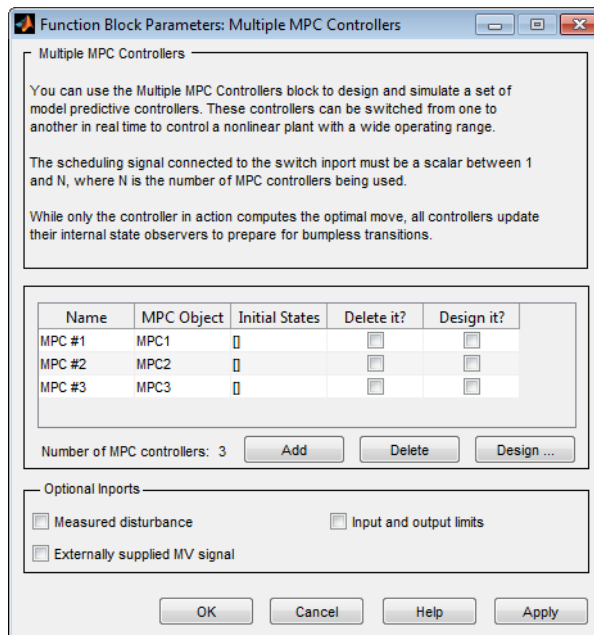
In order to designate which one of the three controllers is active at each time instant, we send the `Multi MPC Controllers` block a switching signal (connected to its `switch` input port). If it is 1, `MPC1` is active. If it is 2, `MPC2` is active, and so on.

In the diagram, `Switch 1` and `Switch 2` perform the controller selection function as follows:

- If the reactor concentration is 8 kmol/m^3 or greater, `Switch 1` sends the constant 1 to its output. Otherwise it sends the constant 2.

- If the reactor concentration is 3 kmol/m^3 or greater, `Switch 2` passes through the signal coming from `Switch 1` (either 1 or 2). Otherwise is sends the constant 3.

Thus, each controller handles a particular composition range. The simulation begins with the reactor at an initial steady state of 311K and 8.57 kmol/m^3. The feed concentration is 10 kmol/m^3 so this is a conversion of about 15%, which is low. The control objective is to transition smoothly to 80% conversion with the reactor concentration at 2 kmol/m^3. The simulation will start with MPC1 active, transition to MPC2, and end with MPC3.

We decide to design the controllers around linear models derived at the following three reactor compositions (and the corresponding steady-state temperature): 8.5, 5.5, and 2 kmol/m^3.

In practice, you would probably obtain the three models from data. This example linearizes the nonlinear model at the above three conditions (for details see "Using Simulink to Develop LTI Models" in the Getting Started Guide).

Note As shown later, we need to retain at the unmeasured plant inputs in the model. This prevents us from using the Model Predictive Control Toolbox automatic linearization feature. In the current toolbox, the automatic linearization feature can linearize with respect to manipulated variable and measured disturbance inputs only.

The following code obtains the linear models and designs the three controllers

```
[sys, xp] = CSTR_INOUT([],[],[],'sizes');
up = [10 298.15 298.15]';
yp = xp;
Ts = 1;
Nc = 3;
Controllers = cell(1,3);
Concentrations = [8.5 5.5 2];
Y = yp;
```

```
for i = 1:Nc
    clear Model
    Y(2) = Concentrations(i);
    [X,U,Y,DX]=trim('CSTR_INOUT',xp(:),up(:),Y(:),[],[1,2]',2)
 [a,b,c,d]=linmod('CSTR_INOUT', X, U );
    Plant = ss(a,b,c,d);
    Plant.InputGroup.MV = 3;
    Plant.InputGroup.UD = [1,2];
    Model.Plant = Plant;
 Model.Nominal.U = [0; 0; up(3)];
    Model.Nominal.X = xp;
    Model.Nominal.Y = yp;
    MPCobj = mpc(Model, Ts);
    MPCobj.Weight.OV = [0 1];
    D = ss(getindist(MPCobj));
    D.b = D.b*10;
    set(D,'InputName',[],'OutputName',[],'InputGroup',[], ...
  'OutputGroup',[]);
    setindist(MPCobj, 'model', D);
    Controllers{i} = MPCobj;
end
MPC1 = Controllers{1};
MPC2 = Controllers{2};
MPC3 = Controllers{3}
```

The key points regarding the designs are as follows:

- All three controllers use the same nominal condition, the values of the plant inputs and outputs at the initial steady-state. Exception: all unmeasured disturbance inputs must have zero nominal values.

- Each controller employs a different prediction model. The model structure is the same in each case (input and outputs are identical in number and type) but each model represents a particular steady-state reactor composition.

- It turns out that the MPC2 plant model obtained at 5 kmol/m^3 is open-loop unstable. We must use a model structure that promotes a stable Kalman state estimator. If we include the unmeasured disturbance inputs in the prediction model, the default estimator assumes integrated white noise at each such input, which produces a stable estimator in this case.

- The default estimator signal-to-noise settings are inappropriate, however. If you use them and monitor the state estimates (not shown), the internally estimated temperature and composition can be far from the measured values. To overcome this, we increase the signal-to-noise ratio in each disturbance channel. See the use of `getindist` and `setindist` above. The default signal to noise is being increased by a factor of 10.

- We are using a zero weight on the measured temperature. See the above discussion of control objectives for the rationale.
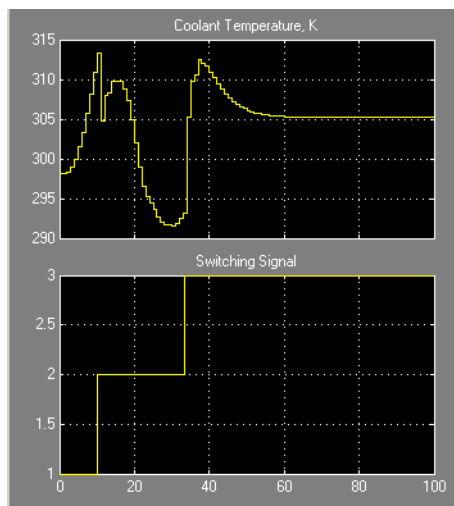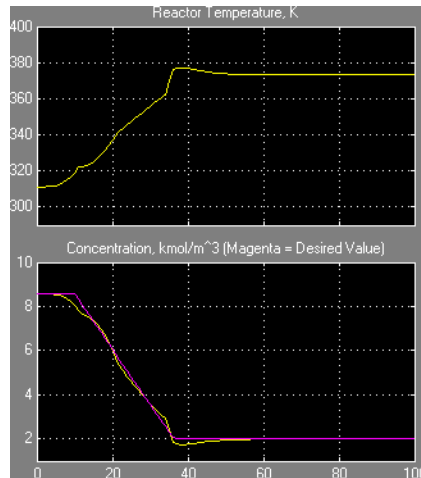
The above plots show the simulation results. The Multi MPC Controller block uses the three controllers sequentially as expected (see the switching signal). Tracking of the concentration setpoint is excellent and the reactor temperature is also controlled well.

To achieve this, the control system starts by increasing the coolant temperature, causing the reaction rate to increase. Once the reaction has achieved a high rate, it generates substantial heat and the coolant temperature must decrease to keep the reactor temperature under control. As the reactor concentration depletes, the reaction rate slows and the control system must again raise the coolant temperature, finally settling at 305 K, about 7 K above the initial condition.

For comparison the plots below show the results for the same scenario if we force MPC3 to be active for the entire simulation. The CSTR eventually stabilizes at the desired steady-state but both the reactor temperature and composition exhibit large excursions away from the desired conditions.
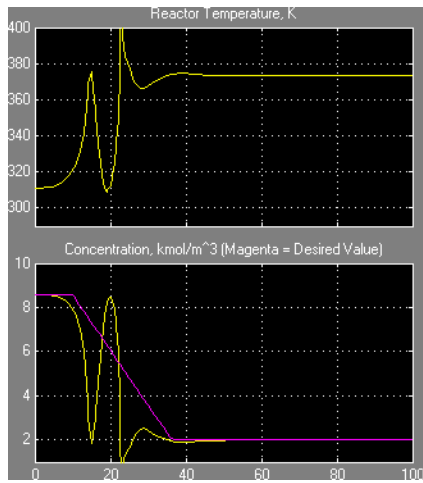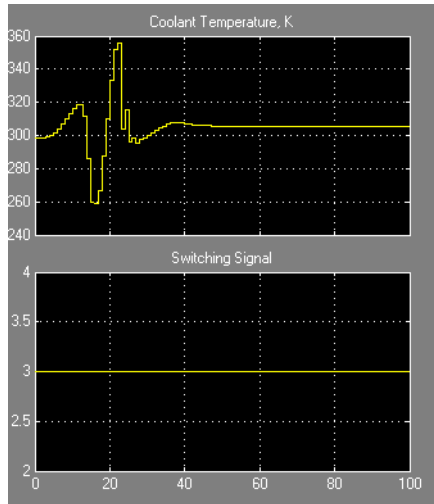
Using Custom Constraints in Blending Process

In this section...
"About the Blending Process" on page 4-57
"MPC Controller with Custom Input/Output Constraints" on page 4-58

About the Blending Process

A continuous blending process combines three feeds in a well-mixed container to produce a blend having desired properties. The dimensionless governing equations are:

$$\frac{dv}{d\tau} = \sum_{i=1}^{3} \phi_i - \phi$$

$$V \frac{d\gamma_j}{d\tau} = \sum_{i=1}^{3} \left(\gamma_{ij} - \gamma_j \right) \phi_i$$

where V is the mixture inventory (in the container), ϕ_i is the flow rate of the ith feed, ϕ is the demand, i.e., the rate at which the blend is being removed from inventory, γ_{ij} is the concentration of constituent j in feed i, γ_j is the concentration of j in the blend, and τ is time. In this example, there are two important constituents, $j = 1$ and 2.

The control objectives are targets for the blend's two constituent concentrations and the mixture inventory. The challenge is that the demand ϕ and feed compositions γ_{ij} vary. The inventory, blend compositions, and demand are measured, but the feed compositions are unmeasured.

At the nominal operating condition:

- Feed 1 ϕ_1 (mostly constituent 1) is 80% of the total inflow ϕ.

- Feed 2 ϕ_2 (mostly constituent 2) is 20%.

- Feed 3 ϕ_3 (pure constituent 1) is not used.

The process design allows manipulation of the total feed entering the mixing chamber and the individual rates of feeds 2 and 3. In other words, the rate of feed 1 is:

$$\varphi_1 = \varphi_T - \varphi_2 - \varphi_3$$

Each feed has limited availability:

$$0 \le \varphi_i \le \varphi_{i,\max}$$

The equations are normalized such that—at the nominal steady state—the mean residence time in the mixing container is $\tau = 1$. The target inventory is $V = 1$, and the target blend composition is $y_1 = y_2 = 1$.

The constraints $\varphi_{i,\max} = 0.8$ is imposed by an upstream process and $\varphi_{2,\max} = \varphi_{3,\max} = 0.6$ is imposed by physical limits.

MPC Controller with Custom Input/Output Constraints

1 Open the Simulink model `mpc_blendingprocess`:
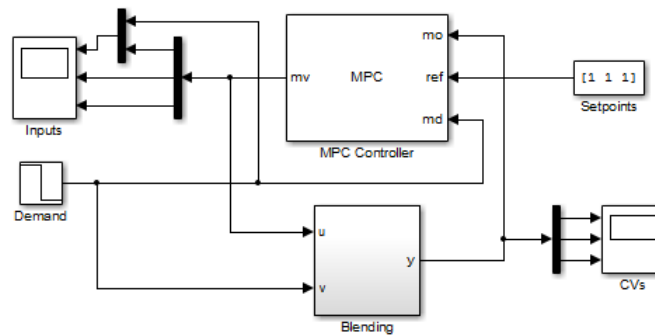
In the model, an MPC controller controls the blending process. The block labeled `Blending` incorporates the previously described model equations and includes unmeasured (step) disturbances in the feed compositions.
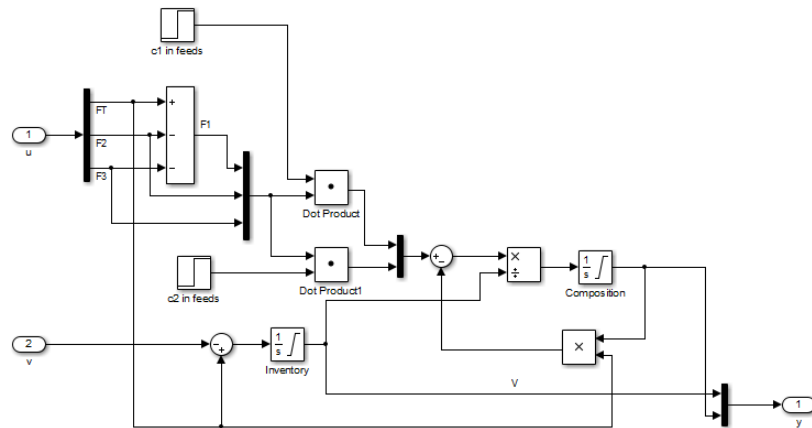
Signal *u* represents controller adjustments, i.e.,

$$u = [\varphi_T \ \varphi_2 \ \varphi_3].$$

Signal *v* represents the demand, φ which is a measured disturbance. The operator can vary the demand, and the resulting signal goes to the process and controller.

Consider the following scenario:

- At $\tau = 0$, the process is operating at its nominal steady state.

- At $\tau = 0.5$, the demand decreases from $\varphi = 1$ to $\varphi = 0.9$.

- At $\tau = 1$, there is a large increase in the concentration of constituent 1 in feed 1 γ_{11} from 1.17 to 2.17.

The plant is mildly nonlinear. You can derive a linear model at the nominal steady state. This approach is quite accurate unless the (unmeasured) feed compositions change. If the change is sufficiently large, the steady-state gains of the nonlinear process change sign and the closed-loop system can become unstable.

2 Define a linear model.

```
% Create a linear approximation -- a state-space model based on the nominal
% operating point:
```

```
ni = 3;                        % number of feed streams
nc = 2;                        % number of components
Fin_nom = [1.6, 0.4, 0];    % Nominal flow rate for the ith feed stream
F_nom  = sum(Fin_nom);       % Nominal flow rate for the exit stream (demand)
cin_nom = [0.7 0.2 0.8       % Nominal composition for jth constituent in the ith feed flow
           0.3 0.8   0];
cout_nom = cin_nom*Fin_nom'/F_nom;  % Nominal product composition

% Normalize the linear model such that the target demand is 1 and the
% product composition is 1:
fin_nom = Fin_nom/F_nom;
gij = [cin_nom(1,:)/cout_nom(1)
       cin_nom(2,:)/cout_nom(2)];

% Create the state-space model with feed flows |[F1, F2, F3]| as MVs:
A = [ zeros(1,nc+1)
      zeros(nc,1) -eye(nc)];
Bu = [ones(1,ni)
       gij-1];
% Change MV definition to [FT, F2, F3] where F1 = FT - F2 - F3
Bu = [Bu(:,1), Bu(:,2)-Bu(:,1), Bu(:,3)-Bu(:,1)];
% Add the blend demand as the 4th model input, a measured disturbance
Bv = [-1
      zeros(nc,1)];
B = [Bu Bv];
% All the states (inventory and compositions) are measurable
C = eye(nc+1);
% No direct feed-through term
D = zeros(nc+1,ni+1);
% Construct the plant model
Model = ss(A, B, C, D);
Model.InputName = {'F_T','F_2','F_3','F'};
Model.InputGroup.MV = 1:3;
Model.InputGroup.MD = 4;
Model.OutputName = {'V','c_1','c_2'};
```

3 Specify an MPC controller.

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

```
Ts = 0.1;
p=10;
m=3;
MPCobj = mpc(Model, Ts, p, m);

% The outputs are the inventory |y(1)| and the constituent concentrations
% |y(2)| and |y(3)|.  Specify nominal values of unity after normalization:
MPCobj.Model.Nominal.Y = [1 1 1];

% The manipulated variables are |u1 = FT|, |u2 = F2|, |u3 = F3|.  Specify
% nominal values after normalization:
MPCobj.Model.Nominal.U = [1 fin_nom(2) fin_nom(3) 1];

% Specify output tuning weights. Larger weights are assigned to the first
% two outputs because we pay more attention to controlling the inventory
% and composition of the first blending material:
MPCobj.Weights.OV = [1 1 0.5];

% Specify the hard bounds (physical limits) on the manipulated variables:
umin = [0 0 0];
umax = [2 0.6 0.6];
for i = 1:3
   MPCobj.MV(i).Min = umin(i);
   MPCobj.MV(i).Max = umax(i);
   MPCobj.MV(i).RateMin = -0.1;
   MPCobj.MV(i).RateMax =  0.1;
end
```

The total feed rate and the rates of feed 2 and feed 3 have upper bounds. Feed 1 also has an upper bound, determined by the upstream unit supplying it. Under normal conditions, the plant operates far from these bounds but for the scenario outlined previously, the controller must reduce the rate of feed 1 drastically, as it is bringing in excess constituent 1. To do this, the controller must increase the rates of feeds 2 and 3 (keeping the total feed rate close to the demand rate to maintain the target inventory.)

4 Specify constraints.

Given the specified bounds on the feed 2 and 3 rates (= 0.6), it is possible that their sum could be as much as 1.2. Because the total feed rate is of

order 0.9 to 1.0, the controller can request a physically impossible condition in which the sum of feeds 2 and 3 exceeds the total feed rate. This implies a negative feed 1 rate.

The constraint

$$0 \le \varphi_1 = \varphi_{T-\varphi_2} - \varphi_3 \le 0.8$$

prevents the controller from requesting an unrealistic φ_1 value.

To specify this constraint, enter:

```
E = [-1 1 1; 1 -1 -1];

% No outputs are specified in the mixed constraints, so set their
% coefficients to zero:
F = zeros(2,3);

% Specify vector g in E*u + F*y <= g:
g = [0; 0.8];

% Specify that both constraints are hard (ECR = 0):
v = zeros(2,1);

% Specify zero coefficients for the measured disturbance:
h = zeros(2,1);

% Include the mixed constraints in the controller object:
setconstraint(MPCobj, E, F, g, v, h);
```

`v = zeros(2,1)` defines hard constraints, which are reasonable here because the constraints involve manipulated variables only. If the constraints involved a mixture of input and output variables, use soft constraints.

5 Simulate the model and plot the input and output signals.

```
figure
plot(MVs.time,[MVs.signals(1).values(:,2), ...
    (MVs.signals(2).values + MVs.signals(3).values), ...
    (MVs.signals(1).values(:,2)-MVs.signals(2).values-MVs.signals(3).values)])
```

```
grid
legend('FT','F2+F3','F1')
```

The plot shows the evolution of the total feed rate (blue curve) and the sum of feeds 2 and 3 (green curve). They coincide between $\tau = 1.7$ and $\tau = 2.2$.
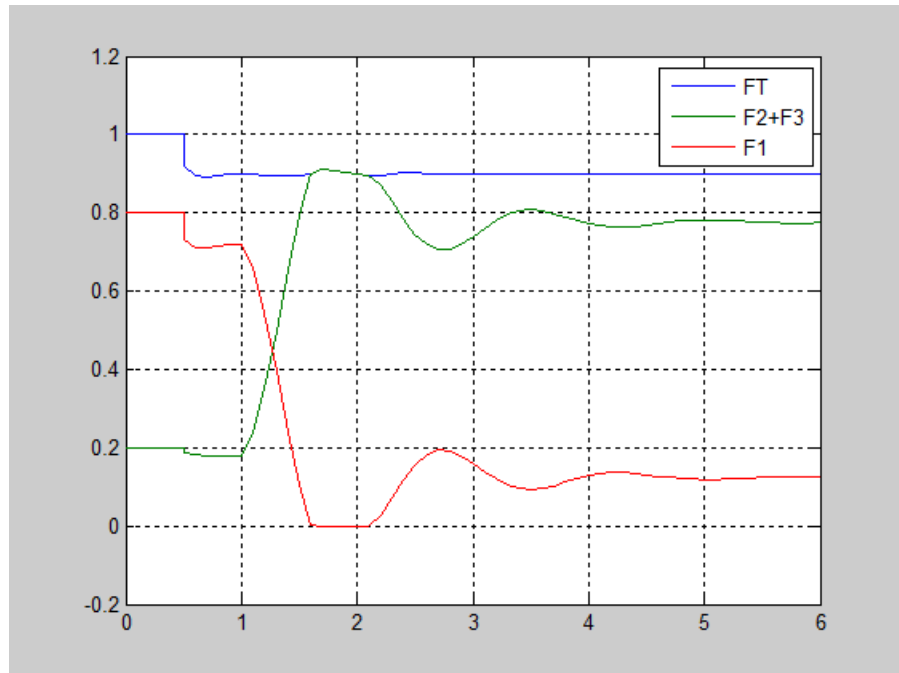
If the custom input constraints had not been included, the controller would have requested a negative feed 1 rate during this period, as shown in by the red curve.

The controller maintains the inventory very close to its setpoint, but the severe disturbance in the feed composition causes a prediction error and a large disturbance in the blend composition (especially for constituent 1). Despite this, the controller recovers and drives the blend composition back to its setpoint, as shown in the following output of the CVs scope.
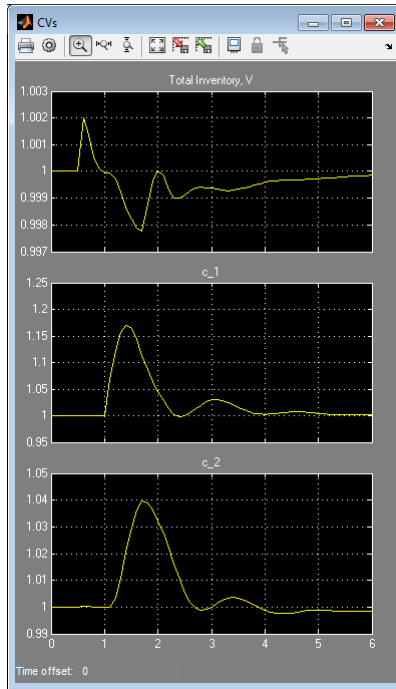
Related Examples

- MPC Control with Constraints on a Combination of Input and Output Signals

- MPC Control of a Nonlinear Blending Process

More About

"Custom Constraints on Inputs and Outputs" on page 2-11

Refine Controller Tuning Weights Using the Tuning Advisor

The Tuning Advisor can help you to refine controller tuning weights for better performance. It also provides a quantitative performance measurement.

You can access the Tuning Advisor from the **Scenarios** node in the Control and Estimation Tools Manager. Before you use the Advisor, choose the controller horizons and sampling period, specify constraints, and select a disturbance estimator (if the default estimator is inappropriate). The Advisor does not provide help with these parameters.

The example considered here is a plant with four controlled outputs and four manipulated variables. There are no measured disturbances and the unmeasured disturbances are unmodeled.

After starting the design tool and importing the plant model, G, which becomes the controller design basis, we accept the default values for all controller parameters. We also load a second plant model, G_p, in which all parameters of G have been perturbed randomly with a standard deviation of 5%.

The scenario shown in the previous figure specifies the controller based on G and the plant G_p. In other words, it tests the controllers robustness with respect to plant-model mismatch. It also defines a series of setpoint changes and disturbances.

Clicking **Tuning Advisor** opens the MPC Tuning Advisor window. In the Tuning Advisor window, we specify the following settings:

- Select the IAE performance function (an arbitrary choice for illustration only).

- Set all input performance weights to zero because the application does not have input targets.

- Set all input rate performance weights to zero because the application has no cost for manipulated variable movement.

- Leave the output performance weights at their default values (unity) because all controller outputs are of roughly equal magnitude and the application gives equal priority to the tracking of all four setpoints.

- Click **Baseline**.

- Click **Analyze**.

Scenario in design: **Scenario1**		Controller in design: **MPC1**		Select a performance function: IAE ▼

Output Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
Out1	1	0.8433	Decrease	1
Out2	1	-2.843	Increase	1
Out3	1	-0.6738	Increase	1
Out4	1	2.769	Decrease	1

Input Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-0.001372	Increase	0
In2	0	-0.0001561	Increase	0
In3	0	-0.002093	Increase	0
In4	0	-9.669e-005	Increase	0

Input Rate Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-2.308	Increase	0.1
In2	0	0.2683	Decrease	0.1
In3	0	-1.368	Increase	0.1
In4	0	2.446	Decrease	0.1

Baseline	Baseline Performance: **26.69**	Analyze	Current Tuning Performance: **26.69**

☑ Tuning Advisor is Modal | Restore Baseline Weights | Update Controller in MPC Tool | Close | Help

The Tuning Advisor resembles the previous figure. The sensitivity values indicate that a decrease in the Out4 weight or an increase in the Out2 weight would have the most impact. In general, however, the output tuning weights should reflect the setpoint tracking priorities and it's preferable to adjust the input rate tuning weights.

Sensitivities for **Input Rate Weights** In1 and In4 are of roughly equal magnitude but the In4 suggestion is a decrease and this weight is already near its lower bound of zero. Thus, we focus on the In1 weight.

The next figure shows the Advisor after the In1 weight has been increased in several steps from 0.1 to 4. Performance has improved by nearly 20% relative to the baseline. Sensitivities indicate that further adjustments to in input rate tuning weights will have little impact.

Scenario in design: **Scenario1**   Controller in design: **MPC1**   Select a performance function: IAE ▼

Output Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
Out1	1	0.7311	Decrease	1
Out2	1	-0.6302	Increase	1
Out3	1	0.4713	Decrease	1
Out4	1	-0.1045	Increase	1

Input Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-6.332e-005	Increase	0
In2	0	-3.406e-005	Increase	0
In3	0	-0.000639	Increase	0
In4	0	-0.0001196	Increase	0

Input Rate Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-0.1172	Increase	4
In2	0	-0.01223	Increase	0.1
In3	0	0.2755	Decrease	0.1
In4	0	-0.2501	Increase	0.1

Baseline   Baseline Performance: **26.69**   Analyze   Current Tuning Performance: **22.58**

☑ Tuning Advisor is Modal   Restore Baseline Weights   Update Controller in MPC Tool   Close   Help

At this point, we can consider adjusting the output tuning weights. It is possible that an attempt to control a particular output might be causing upsets in other outputs (because of model error).

The next figure shows the Tuning Advisor after additional adjustments. At this point, some sensitivities are still rather large, but a small change in the indicated tuning weight causes the sensitivity to change sign. Therefore, futher progress will be difficult.

Scenario in design: **Scenario1** Controller in design: **MPC1** Select a performance function: IAE ▼

Output Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
Out1	1	1.295	Decrease	0.6
Out2	1	-0.006774	Increase	2
Out3	1	-0.1691	Increase	0.3
Out4	1	0.2464	Decrease	1

Input Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-0.0001785	Increase	0
In2	0	0.0001443	Decrease	0
In3	0	0.0004793	Decrease	0
In4	0	0.0005932	Decrease	0

Input Rate Weights

Name	Performance Weight	Sensitivity	Tuning Direction	Current Tuning
In1	0	-0.1493	Increase	4
In2	0	-0.2224	Increase	0.1
In3	0	0.6705	Decrease	0.1
In4	0	-0.4065	Increase	1

Baseline   Baseline Performance: **26.69**   Analyze   Current Tuning Performance: **20.14**

☑ Tuning Advisor is Modal   Restore Baseline Weights   Update Controller in MPC Tool   Close   Help

Overall, we have improved the performance by $(26.69 - 20.14)/26.69$ which is more than 20%.

Providing LQR Performance Using Terminal Penalty

This example, from Scokaert and Rawlings [1], shows how to make a finite-horizon Model Predictive Controller equivalent to an infinite-horizon linear quadratic regulator (LQR).

The standard MPC cost function is similar to that used in an LQR controller with output weighting, as shown in the following equation:

$$J(u) = \sum_{i=1}^{\infty} y(k+i)^T Q y(k+i) + u(k+i-1)^T R u(k+i-1)$$

(4-1)

The LQR and MPC cost functions differ in the following ways:

- The LQR cost function forces y and u towards zero whereas the MPC cost function forces y and u toward nonzero setpoints.

  You can shift the MPC prediction model's origin to eliminate this difference and achieve zero setpoints at nominal condition.

- The LQR cost function uses an infinite prediction horizon in which the manipulated variable changes at each sampling instant. In the standard MPC cost function, the horizon length is p, and the manipulated variable changes m times, where m is the control horizon.

The two cost functions are equivalent if the MPC cost function is:

$$J(u) = \sum_{i=1}^{p-1} y(k+i)^T Q y(k+i) + u(k+i-1)^T R u(k+i-1) + x(k+p)^T Q_p x(k+p)$$

(4-2)

where Q_p is a penalty applied at the last (i.e., terminal) prediction horizon step, and the prediction and control horizons are equal, i.e., $p = m$. The required Q_p is the Ricatti matrix that you can calculate using the Control System Toolbox `lqr` and `lqry` commands. The value is a positive definite symmetric matrix.

The following procedure shows how to design an unconstrained MPC controller that provides performance equivalent to a LQR controller:

1 Define a plant with one input and two outputs.

The plant is a double-integrator, represented as a state-space model in discrete-time with sampling interval 0.1 seconds.

```
A = [1 0;0.1 1];
B = [0.1; 0.005];
C = eye(2);
D = zeros(2,1);
Ts = 0.1;
Plant = ss(A, B, C, D, Ts);
Plant.InputName = {'u'};
Plant.OutputName = {'x_1', 'x_2'};
```

2 Design a LQR controller with output feedback for the plant.

```
Q = eye(2);
R = 1;
[K, Qp] = lqry(Plant, Q, R);
```

`Q` and `R` are output and input weight matrices, respectively. `Q`p is the Ricatti matrix.

3 Design an MPC controller equivalent to the LQR controller.

To implement Equation 4-2, compute L, the Cholesky decompostion of Q_p, such that $L^TL = Q_p$. Then, define auxiliary unmeasured output variables $y_a(k) = Lx(k)$ such that $y_a^{\mathrm{T}}y_a = x^TQ_px$. For the first p-1 prediction horizon steps, the standard Q and R weights apply to the original u and y, and y_a has a zero penalty. On step p, the original u and y have zero penalties, and y_a has a unity penalty.

a Augment the plant model, and specify the augmented outputs as unmeasured.

```
NewPlant = Plant;
cholP = chol(Qp);
set(NewPlant,'C',[C;cholP],'D',[D;zeros(2,1)],'OutputName', {'x_1', 'x_2', 'Cx_1', 'Cx_2'});
NewPlant.InputGroup.MV = 1;
NewPlant.OutputGroup.MO = [1 2];
NewPlant.OutputGroup.UO = [3 4];
```

b Create an MPC controller with prediction horizon equals control horizon.

```
P = 3;
M = 3;
MPCobj = mpc(NewPlant, Ts, P, M);
```

When there are no constraints, you can use a rather short horizon (in this case, $p \geq 1$ gives identical results).

c Specify weights for manipulated variable (MV) and output variable (OV).

```
ywt = sqrt(diag(Q))';
uwt = sqrt(diag(R))';
MPCobj.Weights.OV = [ywt 0 0];
MPCobj.Weights.MV = uwt;
MPCobj.Weights.MVrate = 1e-6;
```

The two augmented outputs have zero weights during the prediction horizon.

d Specify terminal weights.

To obtain the desired effect, define unity weights for these at the final point in the horizon.

```
U = struct('Weight', uwt);
Y = struct('Weight', [0 0 1 1]);
setterminal(MPCobj, Y, U);
setoutdist(MPCobj,'remove'); % Remove added state estimator
setestim(MPCobj, C); % State estimates = measured values
```

The states are measured so the default MPC state estimator is unnecessary. The setoutdist command removes the state estimator, and the setestim command causes the controller state estimates to equal the measured values. The first two states receive zero weight at the terminal point, and the input weight is unchanged.

4 Compare the control performance of LQR, MPC with terminal weights, and a standard MPC.

a Compute closed-loop response with LQR controller.

```
Tstop = 6;
```

```
x0 = [0.2; 0.2];
% compute closed-loop response with LQR
clsys = feedback(Plant,K);
[yLQR tLQR] = initial(clsys,x0,Tstop);
```

b Compute closed-loop response with MPC with terminal weights.

```
SimOptions = mpcsimopt(MPCobj);
SimOptions.PlantInitialState = x0;
r = zeros(1,4);
[y, t, u] = sim(MPCobj, ceil(Tstop/Ts), r, SimOptions);
Cost = sum(sum(y(:,1:2)*diag(ywt).*y(:,1:2))) + sum(u*diag(uwt).*u);
```

c Compute closed-loop response with standard MPC controller.

```
MPCobjSTD = mpc(Plant, Ts); % Default P = 10; M = 2;
MPCobjSTD.Weights.MV = uwt;
MPCobjSTD.Weights.MVrate = 1e-6;
MPCobjSTD.Weights.OV = ywt;
SimOptions = mpcsimopt(MPCobjSTD);
SimOptions.PlantInitialState = x0;
r = zeros(1,2);
[ySTD, tSTD, uSTD] = sim(MPCobjSTD, ceil(Tstop/Ts), r, SimOptions);
CostSTD = sum(sum(ySTD*diag(ywt).*ySTD)) + sum(uSTD*uwt.*uSTD);
```

d Compare the responses.

```
figure;
h1 = line(tSTD,ySTD,'color','r');
Annotation = get(h1,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');
h2 = line(t,y(:,1:2),'color','b');
Annotation = get(h2,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');
h3 = line(tLQR,yLQR,'color','m','marker','o','linestyle','none');
Annotation = get(h3,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');
xlabel('Time');
ylabel('Plant Outputs');
legend('Standard MPC','MPC with Terminal Weigths','LQR','Location','NorthEast')
```

In the following plot, the MPC with the terminal weights provides faster settling to the origin than the standard MPC. The LQR controller and MPC with terminal weights provide identical control performance.
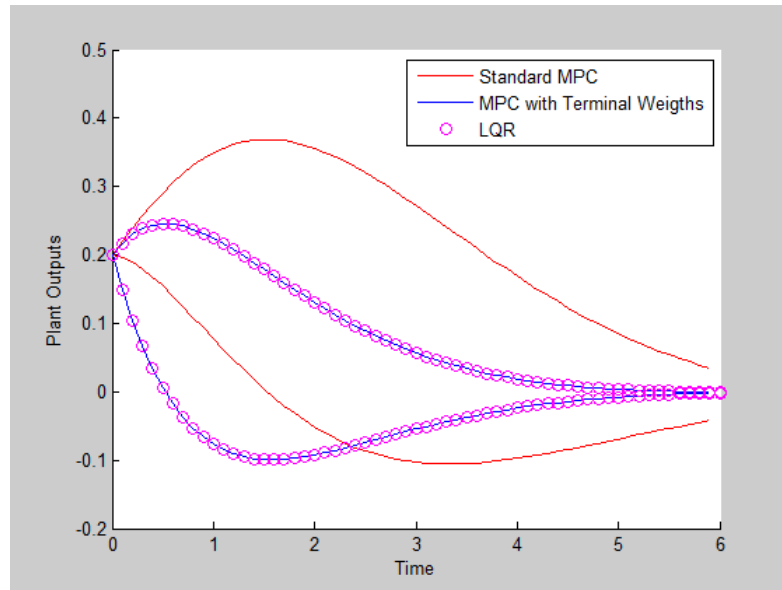
As reported by Scokaert and Rawlings [1], the computed `Cost` value is 2.23, identical to that provided by the LQR controller. The computed `CostSTD` value for the standard MPC is 4.82, more than double compared to `Cost`.

You can improve the standard MPC by retuning. For example, use the same state estimation strategy. If the prediction and control horizons are then increased, it provides essentially the same performance.

This example shows that using a terminal penalty can eliminate the need to tune the MPC prediction and control horizons for the unconstrained case. If your application includes constraints, using a terminal weight is insufficient to guarantee nominal stability. You must also choose appropriate horizons and possibly add terminal constraints. For an in-depth discussion, see Rawlings and Mayne [2].

Although you can design and implement such a controller in Model Predictive Control Toolbox software, you might find designing the standard MPC controller more convenient.

Related Examples

Implementing Infinite-Horizon LQR by Setting Terminal Weights in a Finite-Horizon MPC Formulation

More About

"Terminal Weights and Constraints" on page 2-8

References

[1] Scokaert, P. O. M. and J. B. Rawlings "Constrained linear quadratic regulation" *IEEE Transactions on Automatic Control* (1998), Vol. 43, No. 8, pp. 1163-1169.

Real-Time Control with OPC Toolbox

This example shows how to implement an online model predictive controller application using the OPC client supplied with the OPC Toolbox™.

The example uses the Matrikon™ Simulation OPC server to simulate the behavior of an industrial process on Windows operating system.

Download the Matrikon™ OPC Simulation Server from "www.matrikon.com"

Download and install the server and set it running either as a service or as an application.

This example needs OPC Toolbox™.

```
if ~mpcchecktoolboxinstalled('opc')
    disp('The example needs OPC Toolbox(TM).')
end
```

The example needs OPC Toolbox(TM).

Establish a Connection to the OPC Server

Use OPC Toolbox commands to connect to the Matrikon OPC Simulation Server.

```
if mpcchecktoolboxinstalled('opc')
    % Clear any existing opc connections.
    opcreset
    % Flush the callback persistent variables.
    clear mpcopcPlantStep;
    clear mpcopcMPCStep;
    try
        h = opcda('localhost','Matrikon.OPC.Simulation.1');
        connect(h);
    catch ME
        disp('The Matrikon(TM) OPC Simulation Server must be running on the
        return
    end
end
```

Set up the Plant OPC I/O

In practice the plant would be a physical process, and the OPC tags which define its I/O would already have been created on the OPC server. However, since in this case a simulation OPC server is being used, the plant behavior must be simulated. This is achieved by defining tags for the plant manipulated and measured variables and creating a callback (mpcopcPlantStep) to simulate plant response to changes in the manipulated variables. Two OPC groups are required, one to represent the two manipulated variables to be read by the plant simulator and another to write back the two measured plant outputs storing the results of the plant simulation.

```
if mpcchecktoolboxinstalled('opc')
    % Build an opc group for 2 plant inputs and initialize them to zero.
    plant_read = addgroup(h,'plant_read');
    imv1 = additem(plant_read,'Bucket Brigade.Real8', 'double');
    writeasync(imv1,0);
    imv2 = additem(plant_read,'Bucket Brigade.Real4', 'double');
    writeasync(imv2,0);
    % Build an opc group for plant outputs.
    plant_write = addgroup(h,'plant_write');
    opv1 = additem(plant_write,'Bucket Brigade.Time', 'double');
    opv2 = additem(plant_write,'Bucket Brigade.Money', 'double');
    set(plant_write,'WriteAsyncFcn',[]) % Suppress command line display.
end
```

Specify the MPC Controller Which Will Control the Simulated Plant

Create plant model.

```
plant_model = ss([-.2 -.1; 0 -.05],eye(2,2),eye(2,2),zeros(2,2));
disc_plant_model = c2d(plant_model,1);
% We assume no model mismatch, a control horizon 6 samples and
% prediction horizon 20 samples.
mpcobj = mpc(disc_plant_model,1);
set(mpcobj,'P',20,'ControlHorizon',6);
mpcobj.weights.ManipulatedVariablesRate = [1 1];
% Build an internal MPC object structure so that the MPC object
% is not rebuilt each callback execution.
state = mpcstate(mpcobj);
```

```
y1 = mpcmove(mpcobj,state,[1;1]',[1 1]');

-->The "PredictionHorizon" property of "mpc" object is empty. Trying Predic
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. As
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assumin
-->Integrated white noise added on measured output channel #1.
-->Integrated white noise added on measured output channel #2.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white
```

Build the OPC I/O for the MPC Controller

Build two OPC groups, one to read the two measured plant outputs and the
other to write back the two manipulated variables.

```
if mpcchecktoolboxinstalled('opc')
    % Build an opc group for MPC inputs.
    mpc_read = addgroup(h,'mpc_read');
    impcpv1 = additem(mpc_read,'Bucket Brigade.Time', 'double');
    writeasync(impcpv1,0);
    impcpv2 = additem(mpc_read,'Bucket Brigade.Money', 'double');
    writeasync(impcpv2,0);
    impcref1 = additem(mpc_read,'Bucket Brigade.Int2', 'double');
    writeasync(impcref1,1);
    impcref2 = additem(mpc_read,'Bucket Brigade.Int4', 'double');
    writeasync(impcref2,1);
    % Build an opc group for mpc outputs.
    mpc_write = addgroup(h,'mpc_write');
    additem(mpc_write,'Bucket Brigade.Real8', 'double');
    additem(mpc_write,'Bucket Brigade.Real4', 'double');
    % Suppress command line display.
    set(mpc_write,'WriteAsyncFcn',[]);
end
```

Build OPC Groups to Trigger Execution of the Plant Simulator & Controller

Build two opc groups based on the same external opc timer to trigger
execution of both plant simulation and MPC execution when the contents of
the OPC time tag changes.

```
if mpcchecktoolboxinstalled('opc')
    gtime = addgroup(h,'time');
    time_tag = additem(gtime,'Triangle Waves.Real8');
    set(gtime,'UpdateRate',1);
    gtime.DataChangeFcn = {@mpcopcPlantStep plant_read plant_write disc_pla
    gmpctime = addgroup(h,'mpctime');
    additem(gmpctime,'Triangle Waves.Real8');
    set(gmpctime,'UpdateRate',1);
    gmpctime.DataChangeFcn = {@mpcopcMPCStep mpc_read mpc_write mpcobj};
end
```

Log Data from the Plant Measured Outputs

Log the plant measured outputs from tags 'Bucket Brigade.Money' and
'Bucket Brigade.Money'.

```
if mpcchecktoolboxinstalled('opc')
    set(mpc_read,'RecordsToAcquire',40);
    start(mpc_read);
    while mpc_read.RecordsAcquired < mpc_read.RecordsToAcquire
        pause(3)
        fprintf('Logging data: Record %d / %d',mpc_read.RecordsAcquired,...
            mpc_read.RecordsToAcquire)
    end
    stop(mpc_read);
end
```

Extract and Plot the Logged Data

```
if mpcchecktoolboxinstalled('opc')
    [itemID, value, quality, timeStamp, eventTime] = getdata(mpc_read,'doub
    plot((timeStamp(:,1)-timeStamp(1,1))*24*60*60,value)
    title('Measured Outputs Logged from Tags Bucket Brigade.Time,Bucket Bri
    xlabel('Time (secs)');
end
```

Measured Outputs Logged from Tags Bucket Brigade.Time,Bucket Brigade.Money
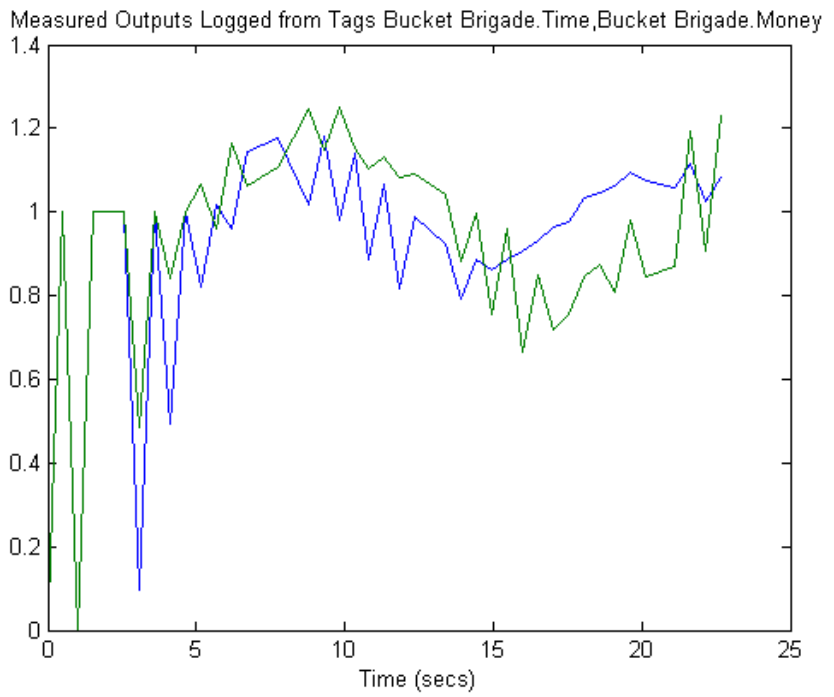
Code Generation with Simulink Coder™

This example shows how to use Simulink Coder to generate real-time code for a MPC Controller block in Simulink.

We illustrate the workflow with a single-input single-output system tracking of a sinusoid reference signal under output constraints.

Required Products

To run this example, Simulink and Simulink Coder™ are required.

```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
if ~mpcchecktoolboxinstalled('simulinkcoder')
    disp('Simulink(R) Coder(TM) is required to run this example.');
    return
end
```

```
Simulink(R) Coder(TM) is required to run this example.
```

Plant Model and MPC Controller Setup

Define plant model.

```
sys = ss(tf([3 1],[1 0.6 1]));
[A,B,C,D] = ssdata(sys);
x0=[0 0]';
```

Define model predictive controller object.

```
Ts = 0.2;   %Sampling time
p = 40;     %Prediction horizon
m = 3;      %Control horizon
MV = struct('Min',-Inf,'Max',Inf,'RateMin',-20,'RateMax',20); % Input const
OV = struct('Min',-0.5,'Max',0.5); % Output constraints
mpccon = mpc(sys,Ts,p,m,[],MV,OV);
```

Simulation Using Simulink

```
Tstop=10;  %Simulation time
open_system('mpc_rtwdemo')
sim('mpc_rtwdemo',Tstop)
```

Build Simulink Model with Simulink Coder

Use "RTWBUILD command from Simulink Coder™ to compile the Simulink model into an executable.

```
close_system('mpc_rtwdemo/Inputs')
close_system('mpc_rtwdemo/Outputs//References')
cwd = pwd;
```

Before starting code generation, we need to change directory to a temp directory so that we have write-permission to generate the relevant target files and the executable.

```
tmpdir = tempname;
mkdir(tmpdir);
cd(tmpdir);
disp('Compiling... Please wait until it finishes.');
rtwbuild('mpc_rtwdemo');
```

Running Executable

If the build finished successfully and if you are able to run the executable (status=0), you should have a data file named mpc_rtwdemo.mat in your temporary directory. You can load this file and compare the u, y and r signals with those generated by simulating the model.

```
disp('Running executable...');
if ispc
    status = system('mpc_rtwdemo');
else
    disp('The example only runs the executable on Windows system.');
end
```

Clean up.

```
cd(cwd)
bdclose('mpc_rtwdemo')
```

Setting Targets for Manipulated Variables

This example shows how to design a model predictive controller for a plant with two inputs and one output with target set-point for a manipulated variable.

Define the Plant to be Controlled

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

N1=[3 1];
D1=[1 2*.3 1];
N2=[2 1];
D2=[1 2*.5 1];
sys=ss(tf({N1,N2},{D1,D2})),'min');
A=sys.a;B=sys.b;C=sys.c;D=sys.d;
x0=[0 0 0 0]';
```

MPC Controller Setup

```
Ts=.4;                      % Sampling time
model=c2d(ss(A,B,C,D),Ts);  % discrete-time prediction model
mpcobj=mpc(model,Ts,20,5);

-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. As
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assumin
```

Change default weights.

```
mpcobj.weights.manipulated=[0.3 0];      % weight difference MV#1 - Target#1
mpcobj.weights.manipulatedrate=[0 0];
mpcobj.weights.output=1;
```

Define input specifications.

```
clear MV
MV(1)=struct('RateMin',-.5,'RateMax',.5);
```

```
MV(2)=struct('RateMin',-.5,'RateMax',.5);
```

The following sets up a target set-point u=2 for the first manipulated variable.

```
MV(1).Target=2; % Input steady-state set-point
mpcobj.MV=MV;
```

Simulation Using Simulink

```
Tstop=40;                    % Simulation time
open_system('mpc_utarget')   % Open Simulink(R) Model
sim('mpc_utarget',Tstop);    % Start Simulation
```

```
-->Integrated white noise added on measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white
```
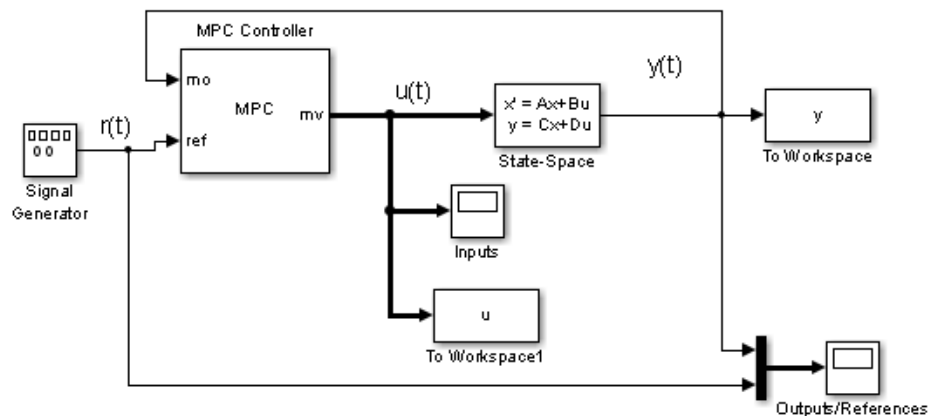

Copyright 1990-2012 The MathWorks, Inc.
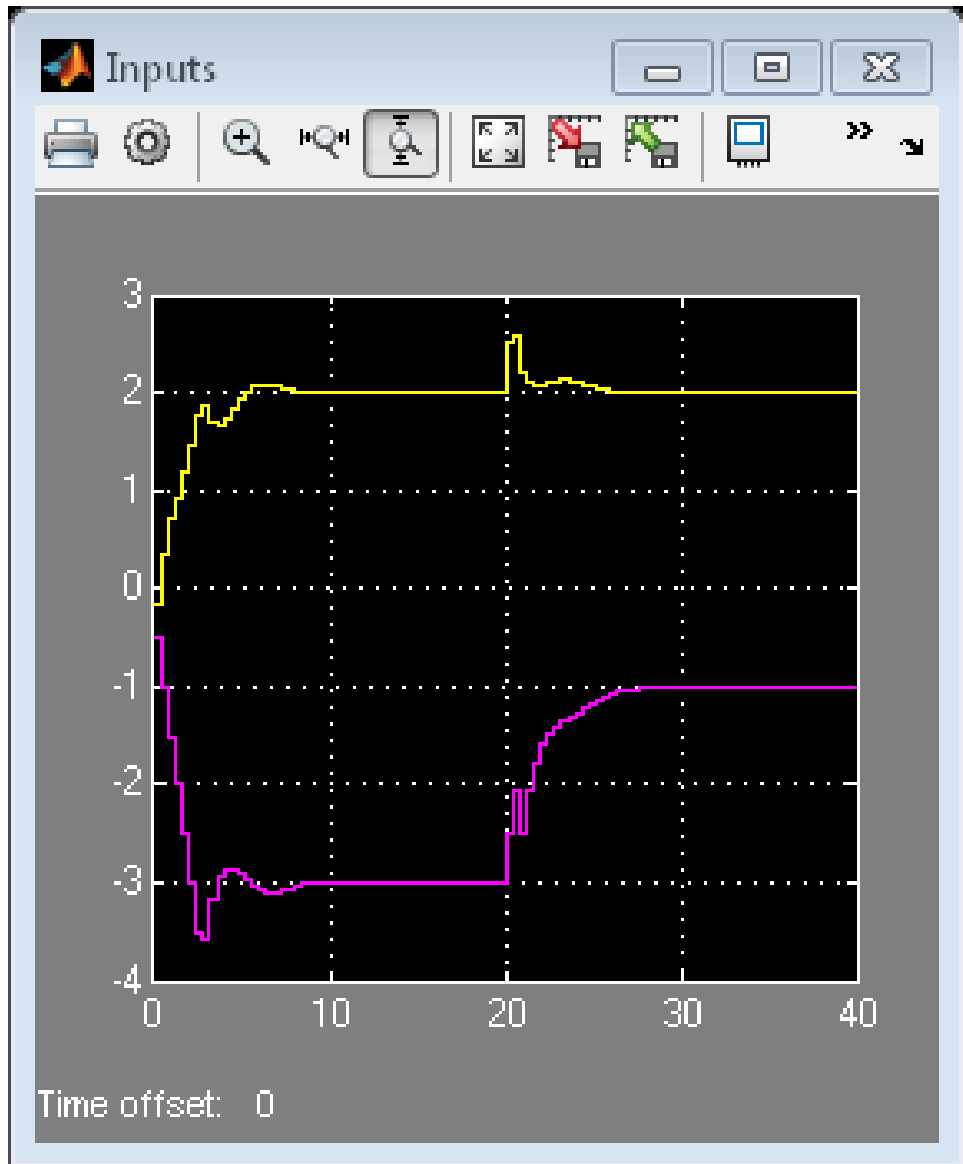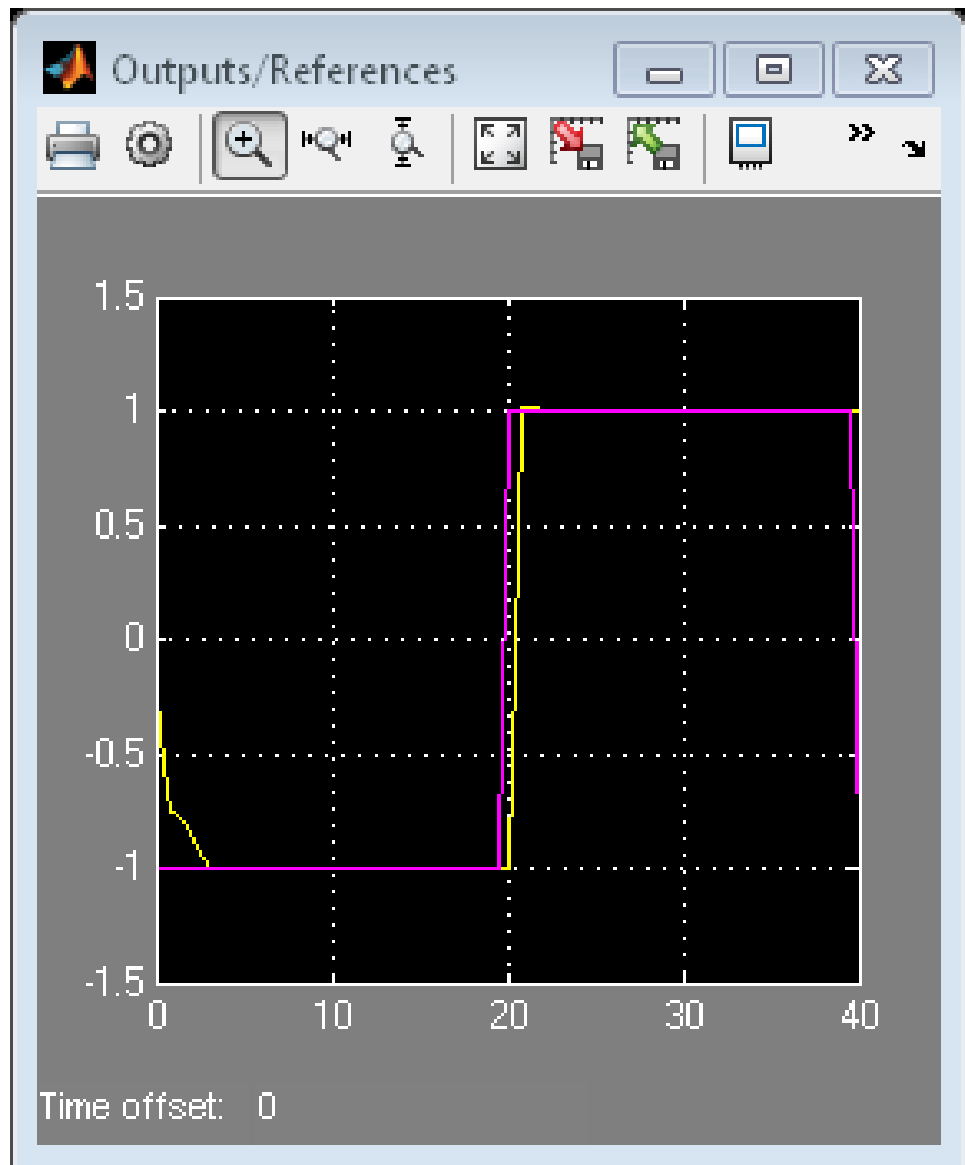
```
bdclose('mpc_utarget')
```

Specifying Alternative Cost Function with Off-Diagonal Weight Matrices

This example shows how to use non-diagonal weight matrices in a model predictive controller.

MPC Controller Setup

We start defining the plant to be controlled.

```
sys=ss(tf({1,1;1,2},{[1 .5 1],[.7 .5 1];[1 .4 2],[1 2]}),'min');
```

Now, setup an MPC controller object.

```
Ts=.1;              % sampling time
model=c2d(sys,Ts);  % prediction model
```

Define prediction and control horizons.

```
p=20;        % prediction horizon
m=2;         % control horizon
```

Let us assume default value for weights and build the MPC object.

```
MPCobj=mpc(model,Ts,p,m);
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. As
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assumin
```

Define constraints on the manipulated variable.

```
clear MV
MV(1)=struct('Min',-3,'Max',3,'RateMin',-100,'RateMax',100);
MV(2)=struct('Min',-2,'Max',2,'RateMin',-100,'RateMax',100);
MPCobj.MV=MV;
```

Define non-diagonal output weight. Note that it is specified inside a cell!

```
OW=[1 -1]'*[1 -1];
```

```
% Non-diagonal output weight, corresponding to ((y1-r1)-(y2-r2))^2
MPCobj.Weights.OutputVariables={OW};
% Non-diagonal input weight, corresponding to (u1-u2)^2
MPCobj.Weights.ManipulatedVariables={0.5*OW};
```

Closed-Loop MPC Simulation Using the Command SIM

```
Tstop=30;                % simulation time

Tf=round(Tstop/Ts);      % number of simulation steps
r=ones(Tf,1)*[1 2];      % reference trajectory
```

Run the closed-loop simulation and plot results.

```
close all
[y,t,u]=sim(MPCobj,Tf,r);
subplot(211)
plot(t,y(:,1)-r(1,1)-y(:,2)+r(1,2));grid
title('(y_1-r_1)-(y_2-r_2)');
subplot(212)
plot(t,u);grid
title('u');
```

```
-->Integrated white noise added on measured output channel #1.
-->Integrated white noise added on measured output channel #2.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white
```

$(y_1 - r_1) - (y_2 - r_2)$

u
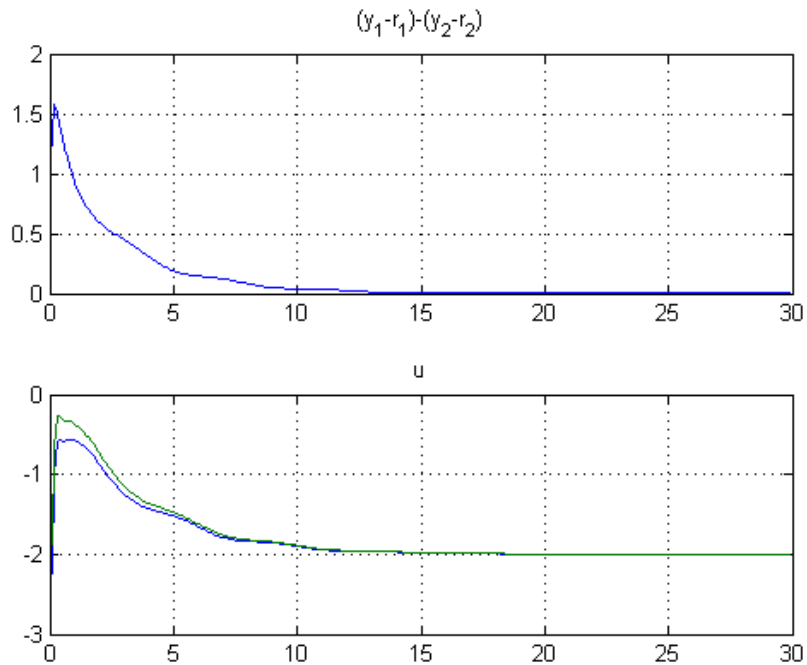
MPC Simulation Using Simulink

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

The continuous-time plant to be controlled has the following state-space realization:

```
[A,B,C,D]=ssdata(sys);
```

Now simulate closed-loop MPC in Simulink.

```
open_system('mpc_weightsdemo');
sim('mpc_weightsdemo',Tstop)
```
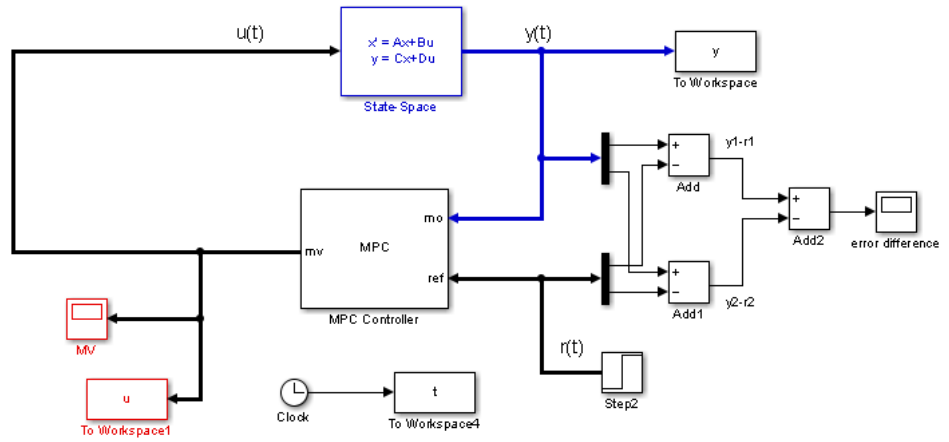
Copyright 1990-2012 The MathWorks, Inc.
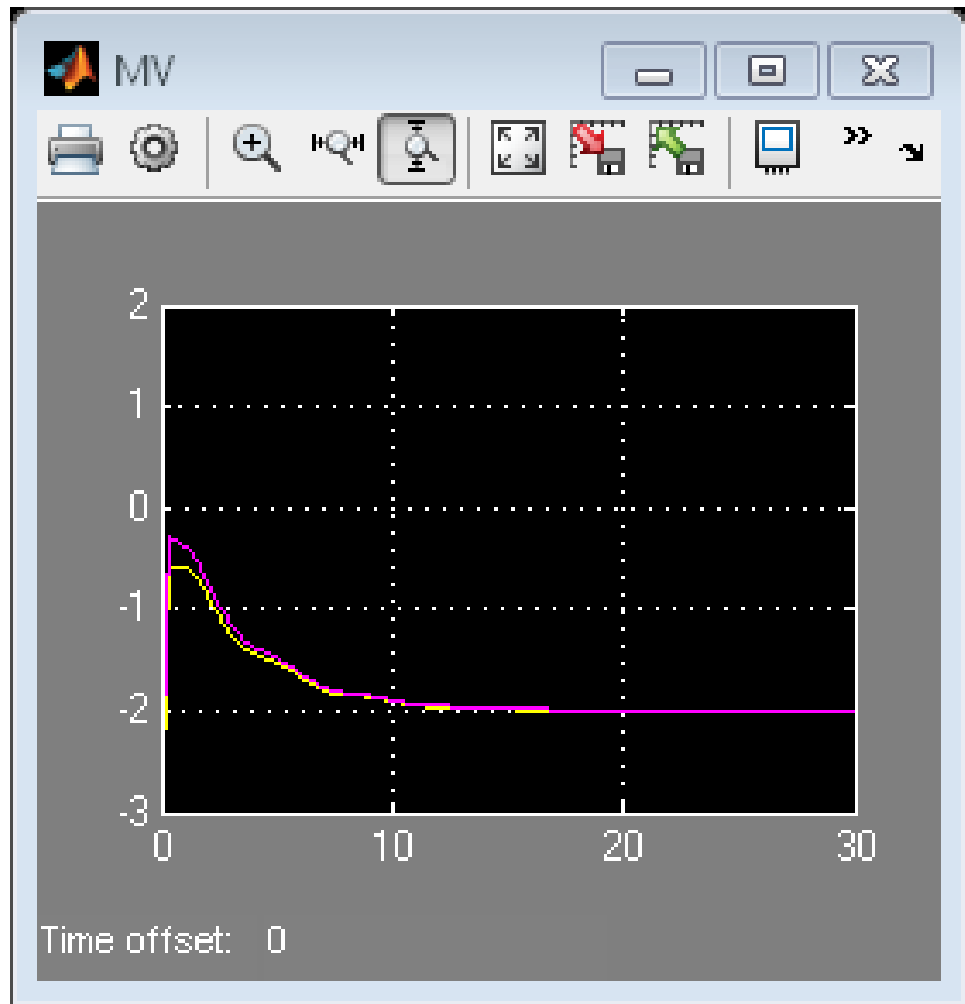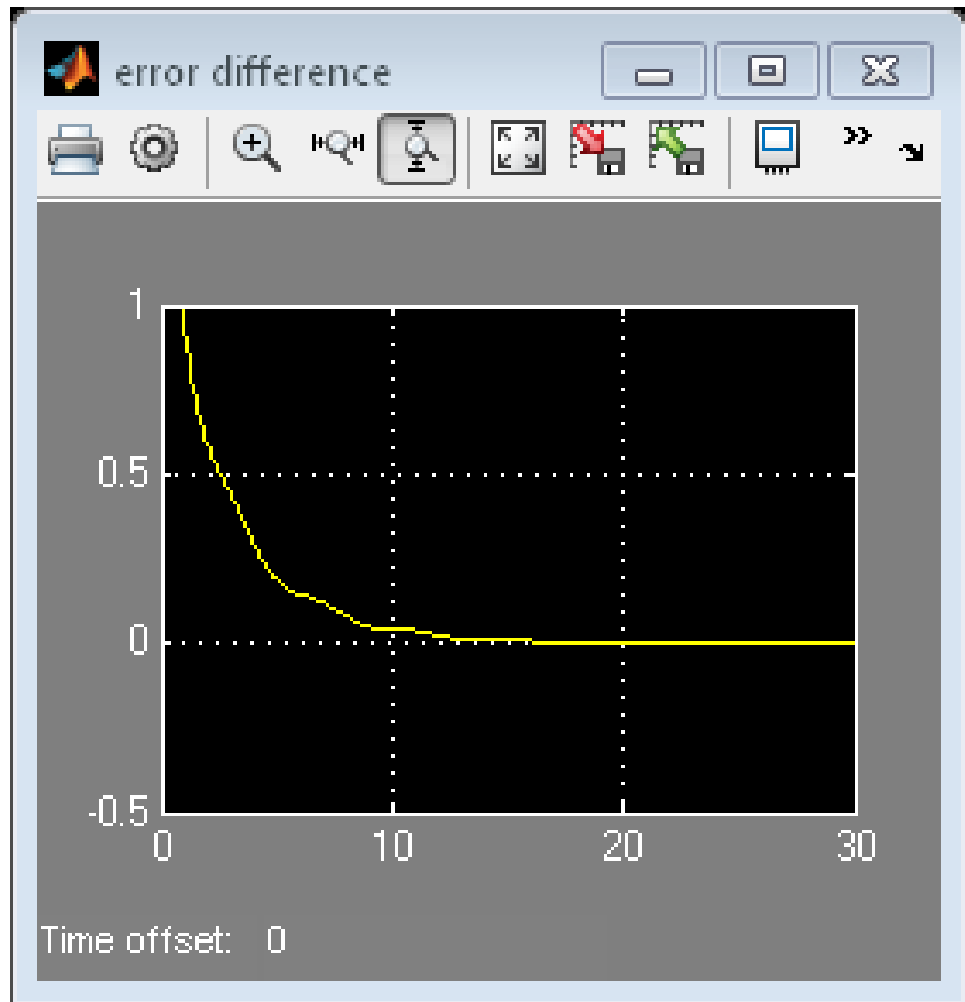
```
bdclose('mpc_weightsdemo');
```

Review Model Predictive Controller Design for Potential Stability and Robustness Issues

This example shows how to use the `review` command to detect potential issues with a model predictive controller design.

The Fuel Gas Blending Process

The example application is a fuel gas blending process. The objective is to blend six gases to obtain a fuel gas, which is then burned to provide process heating. The fuel gas must satisfy three quality standards in order for it to burn reliably and with the expected heat output. The fuel gas header pressure must also be controlled. Thus, there are four controlled output variables. The manipulated variables are the six feed gas flow rates.

Inputs:

```
1. Natural Gas (NG)
2. Reformed Gas (RG)
3. Hydrogen (H2)
4. Nitrogen (N2)
5. Tail Gas 1 (T1)
6. Tail Gas 2 (T2)
```

Outputs:

```
1. High Heating Value (HHV)
2. Wobbe Index (WI)
3. Flame Speed Index (FSI)
4. Header Pressure (P)
```

The fuel gas blending process was studied by Muller et al.: "Modeling, validation, and control of an industrial fuel gas blending system", C.J. Muller, I.K. Craig, N.L. Ricker, J. of Process Control, in press, 2011.

Linear Plant Model

The plant is nonlinear but the basis for the controller design is the following linear predictive model in the state space format, which applies at a typical steady state operating point. Time units are hours.

```
a = diag([-28.6120, -28.6822, -28.5134,  -0.0281, -23.2191, -23.4266, ...
          -22.9377, - 0.0101, -26.4877, -26.7950, -27.2210,  -0.0083, ...
          -23.0890, -23.0062, -22.9349,  -0.0115, -25.8581, -25.6939, ...
          -27.0793,  -0.0117, -22.8975, -22.8233, -21.1142,  -0.0065]);
b = zeros(24,6);
b( 1: 4,1) = [4, 4, 8, 32]';
b( 5: 8,2) = [2, 2, 4, 32]';
b( 9:12,3) = [2, 2, 4, 32]';
b(13:16,4) = [4, 4, 8, 32]';
b(17:20,5) = [2, 2, 4, 32]';
b(21:24,6) = [1, 2, 1, 32]';
c = [diag([ 6.1510,  7.6785, -5.9312, 34.2689]), ...
     diag([-2.2158, -3.1204,  2.6220, 35.3561]), ...
     diag([-2.5223,  1.1480,  7.8136, 35.0376]), ...
     diag([-3.3187, -7.6067, -6.2755, 34.8720]), ...
     diag([-1.6583, -2.0249,  2.5584, 34.7881]), ...
     diag([-1.6807, -1.2217,  1.0492, 35.0297])];
d = zeros(4,6);
Plant = ss(a, b, c, d);
```

By default, all the plant inputs are manipulated variables.

```
Plant.InputName = {'NG', 'RG', 'H2', 'N2', 'T1', 'T2'};
```

By default, all the plant outputs are measured outputs.

```
Plant.OutputName = {'HHV', 'WI', 'FSI', 'P'};
```

Transport delay is added to plant outputs to reflect the delay in the sensors.

```
Plant.OutputDelay = [0.00556  0.0167  0.00556  0];
```

Initial Controller Design

Construct an initial model predictive controller based on design requirements.

Specify sampling time, horizons and steady state values.

The sampling time is that of the sensors (20 seconds). The prediction horizon is approximately equal to the plant settling time (39 intervals). The control

horizon uses four blocked moves that have lengths of 2, 6, 12 and 19 intervals respectively. The nominal operating conditions are non-zero. The output measurement noise is white noise with magnitude of 0.001.

```
MPC_verbosity = mpcverbosity('off'); % Disable MPC message displaying at co
Ts = 20/3600;   % Time units are hours.
Obj = mpc(Plant, Ts, 39, [2, 6, 12, 19]);
Obj.Model.Noise = ss(0.001*eye(4));
Obj.Model.Nominal.Y = [16.5, 25, 43.8, 2100];
Obj.Model.Nominal.U = [1.4170, 0, 2, 0, 0, 26.5829];
```

Specify lower and upper bounds on manipulated variables.

Since all the manipulated variables are flow rates of gas streams, their lower bounds are zero. All the MV constraints are hard (MinECR and MaxECR = 0) by default.

```
MVmin = zeros(1,6);
MVmax = [15, 20, 5, 5, 30, 30];
for i = 1:6
    Obj.MV(i).Min = MVmin(i);
    Obj.MV(i).Max = MVmax(i);
end
```

Specify lower and upper bounds on manipulated variable increments.

The bounds are set large enough to allow full range of movement in one interval. All the MV rate constraints are hard (RateMinECR and RateMaxECR = 0) by default.

```
for i = 1:6
    Obj.MV(i).RateMin = -MVmax(i);
    Obj.MV(i).RateMax =  MVmax(i);
end
```

Specify lower and upper bounds on plant outputs.

All the OV constraints are soft (MinECR and MaxECR = 0) by default.

```
OVmin = [16.5, 25, 39, 2000];
OVmax = [18.0, 27, 46, 2200];
```

```
for i = 1:4
    Obj.OV(i).Min = OVmin(i);
    Obj.OV(i).Max = OVmax(i);
end
```

Specify weights on manipulated variables.

MV weights are specified based on the relative cost of the blended streams. This allows MPC to choose among the six in order to minimize the cost of the blended fuel gas.

```
Obj.Weights.MV = [54.9, 20.5, 0, 5.73, 0, 0];
```

Specify weights on manipulated variable increments.

They are relatively small to allow freedom of movement.

```
Obj.Weights.MVrate = 1.22*ones(1,6);
```

Specify weights on plant outputs.

The OV weights reflect the relative magnitudes of the output variable signals, considering that the predictive model is not scaled. Since the nominal value of the header pressure (the 4th output) is relatively large, set its weight smaller to compensate.

```
Obj.Weights.OV = [82, 65, 57, 16];
```

Using the `review` Command to Improve the Initial Design

Review the initial controller design.

```
review(Obj)
```
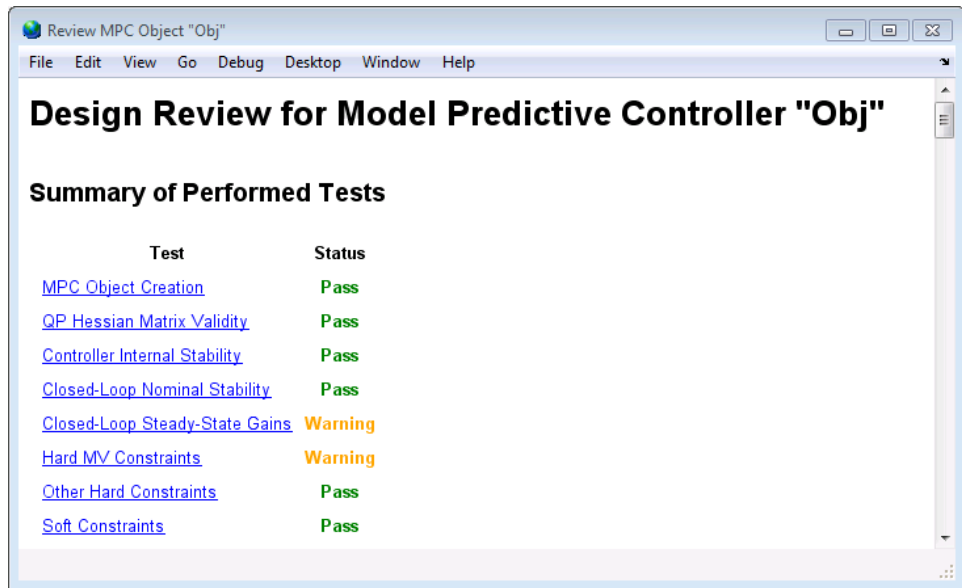
The summary table shown above lists two warnings. The first warning is that the controller won't drive the output variables to their targets at steady state. Click **Closed-Loop Steady-State Gains** to see a list of the non-zero gains.

The first entry in the list shows that adding a sustained disturbance of unit magnitude to the HHV output would cause the HHV to deviate 0.0156 units from its steady-state target, assuming no constraints are active. The second entry shows that a unit disturbance in WI would cause a steady state deviation ("offset") of -0.0469 in HHV, etc.

Since there are six MVs and only four OVs, enough degrees of freedom are available in control system and you might be surprised to see non-zero steady state offset. The root cause is the non-zero MV weights we use to force the plant toward the most economical operating condition. The non-zero steady state offsets are often undesirable but they are acceptable in this case because the primary objective is to minimize the blend cost and the gas quality is allowed to float as long as it stays within the specified limits, even when a process disturbance is present Small offset gain magnitudes indicate that the impact of disturbance would be limited. Moreover, small violations of those limits would be acceptable as well because output constraints are soft.

Review the second warning by clicking **Hard MV Constraints**, which indicates a potential hard-constraint conflict.

Review MPC Object "Obj"

File　Edit　View　Go　Debug　Desktop　Window　Help

Hard MV Constraints

The controller should always satisfy hard bounds on a manipulated variable *OR* its rate-of-change. If you specify both constraint types simultaneously, however, they might conflict during real-time use.

For example, if an event pushes an MV outside a specified hard bound and the hard MV rate bounds are too small, the resulting QP will be *infeasible*.

Avoid such conflicts by specifying hard MV bounds *OR* hard MV rate bounds, but not both. Or if you want to specify both, soften the lower-priority constraint by setting its ECR to a value greater than zero.

Warning: your constraint definitions may conflict. The following table lists potential conflicts for each MV. The tabular entries show the location of each conflict in the prediction horizon and the type of conflict.

MV name	Horizon k	Conflict Type
NG	1	Min & RateMax
NG	1	Max & RateMin
RG	1	Min & RateMax
RG	1	Max & RateMin
H2	1	Min & RateMax
H2	1	Max & RateMin
N2	1	Min & RateMax
N2	1	Max & RateMin
T1	1	Min & RateMax
T1	1	Max & RateMin
T2	1	Min & RateMax
T2	1	Max & RateMin

Return to list of tests

If an external event causes the NG to go far below its specified minimum, the constraint on its rate of increase might make it impossible to return the NG within bounds in one interval. In other words, when you specify both MV.Min and MV.RateMax, the controller would not be able to find an optimal solution if the most recent MV value is less than (MV.Min - MV.RateMax). Similarly, there is a potential conflict when you specify both MV.Max and MV.RateMin.

An MV constraint conflict would be extremely unlikely in the gas blending application, but it's good practice to eliminate the possibility by softening one of the two constraints. Since the MV minimum and maximum values are respected in this example, we soften the increment bounds as follows:

4-101

```
for i = 1:6
    Obj.MV(i).RateMinECR = 0.1;
    Obj.MV(i).RateMaxECR = 0.1;
end
```

Review the new controller design.

```
review(Obj)
```
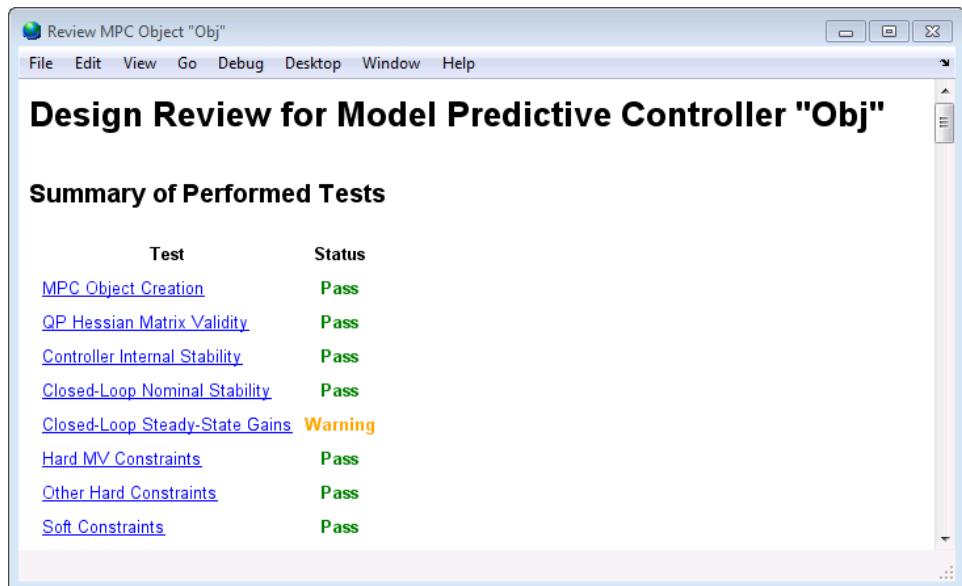

The MV constraint conflict warning has disappeared.

Diagnosing the Impact of Zero Output Weights

Assuming that a new design requirement allows the controlled outputs vary freely within their limits, consider eliminating the weights on them:
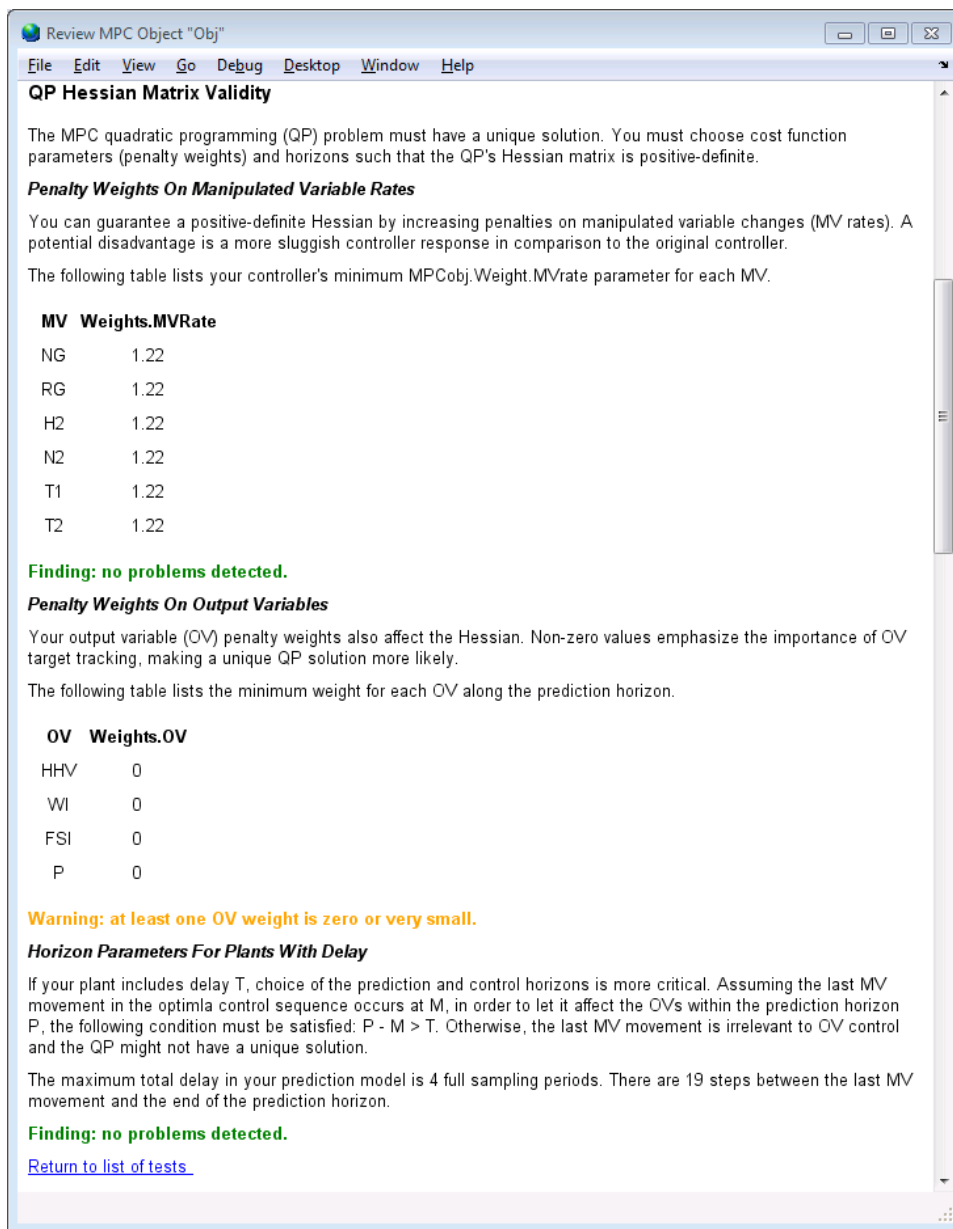
```
Obj.Weights.OV = zeros(1,4);
```

Review the impact of this design change.

```
review(Obj)
```
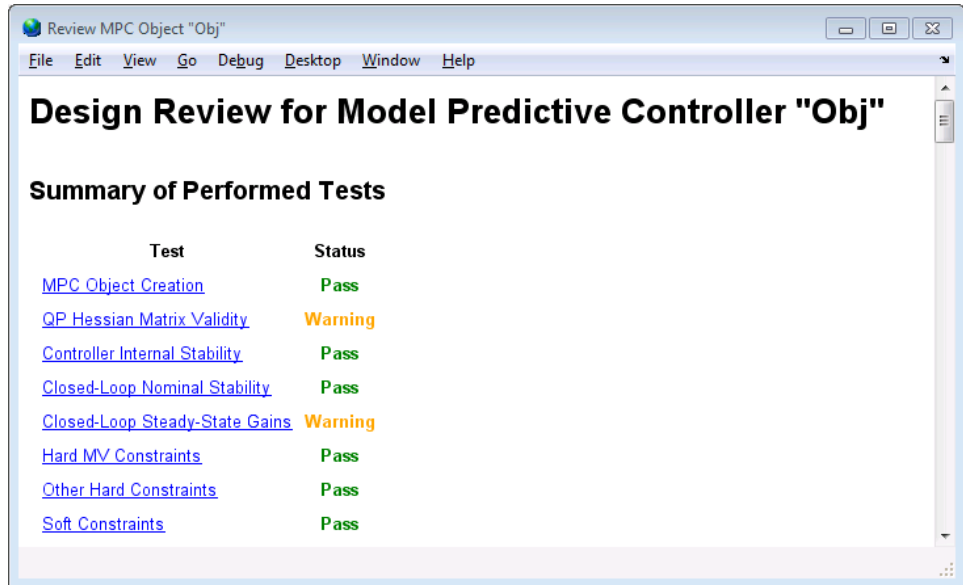

A new warning regarding QP Hessian Matrix Validity appears, in addition to the steady-state gain warning. Click **QP Hessian Matrix Validity** warning to see the details.

Review MPC Object "Obj"

File   Edit   View   Go   Debug   Desktop   Window   Help

QP Hessian Matrix Validity

The MPC quadratic programming (QP) problem must have a unique solution. You must choose cost function parameters (penalty weights) and horizons such that the QP's Hessian matrix is positive-definite.

Penalty Weights On Manipulated Variable Rates

You can guarantee a positive-definite Hessian by increasing penalties on manipulated variable changes (MV rates). A potential disadvantage is a more sluggish controller response in comparison to the original controller.

The following table lists your controller's minimum MPCobj.Weight.MVrate parameter for each MV.

MV	Weights.MVRate
NG	1.22
RG	1.22
H2	1.22
N2	1.22
T1	1.22
T2	1.22

Finding: no problems detected.

Penalty Weights On Output Variables

Your output variable (OV) penalty weights also affect the Hessian. Non-zero values emphasize the importance of OV target tracking, making a unique QP solution more likely.

The following table lists the minimum weight for each OV along the prediction horizon.

OV	Weights.OV
HHV	0
WI	0
FSI	0
P	0

Warning: at least one OV weight is zero or very small.

Horizon Parameters For Plants With Delay

If your plant includes delay T, choice of the prediction and control horizons is more critical. Assuming the last MV movement in the optimla control sequence occurs at M, in order to let it affect the OVs within the prediction horizon P, the following condition must be satisfied: P - M > T. Otherwise, the last MV movement is irrelevant to OV control and the QP might not have a unique solution.

The maximum total delay in your prediction model is 4 full sampling periods. There are 19 steps between the last MV movement and the end of the prediction horizon.

Finding: no problems detected.

Return to list of tests

We see that the review has flagged the zero weights on all four output variables. But since the zero weights come from design requirement and the other Hessian tests indicate that the quadratic programming problem should have a unique solution, this warning can be safely ignored.

Click **Closed-Loop Steady-State Gains** to see the second warning. It shows another consequence of setting the four OV weights to zero. When an OV is not penalized by a weight, any output disturbance added to it will be ignored, passing through with no attenuation.

Since it is a design requirement, non-zero steady state offsets are acceptable provided that MPC is able to hold all the OVs within their specified bounds. It is therefore a good idea to examine how easily the soft OV constraints can be violated when disturbances are present.

Reviewing Soft Constraints

Click **Soft Constraints** to see a list of soft constraints -- in this case an upper and lower bound on each OV.
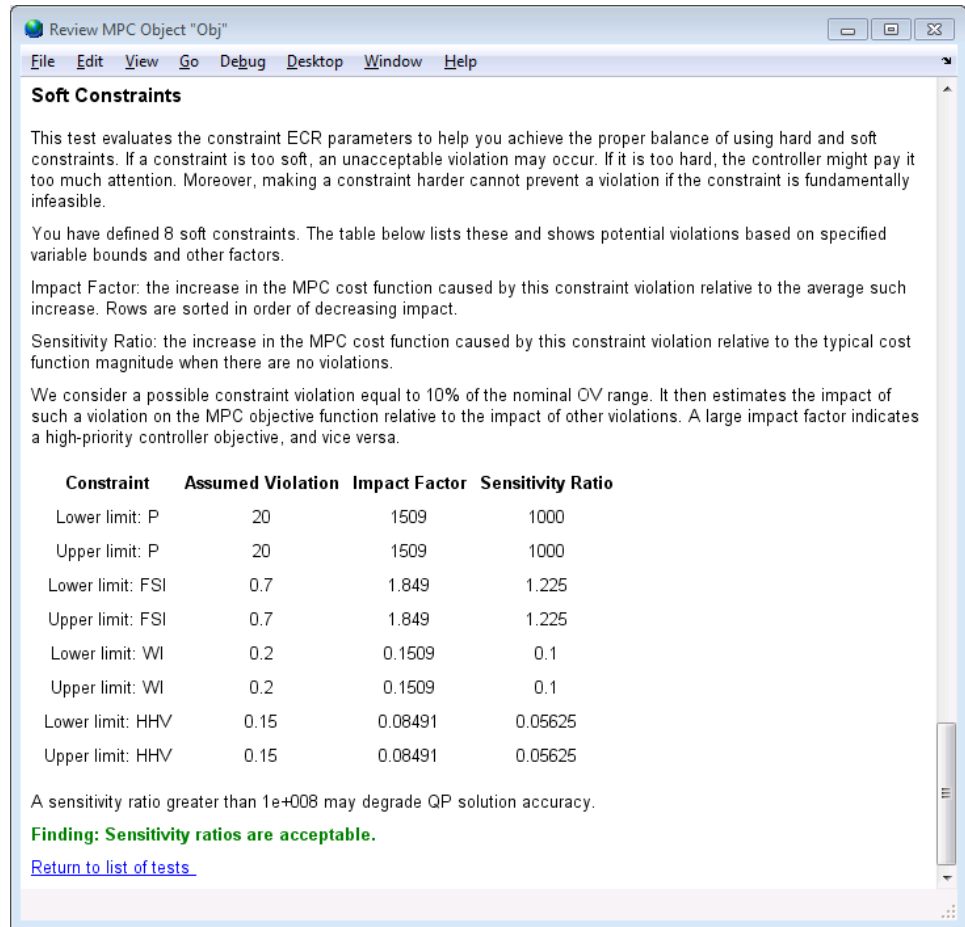
The Impact Factor column shows that using the default MinECR and MaxECR values gives the pressure (P) a much higher priority than the other OVs. If we want the priorities to be more comparable, we should increase the pressure constraint ECR values and adjust the others too. For example, use
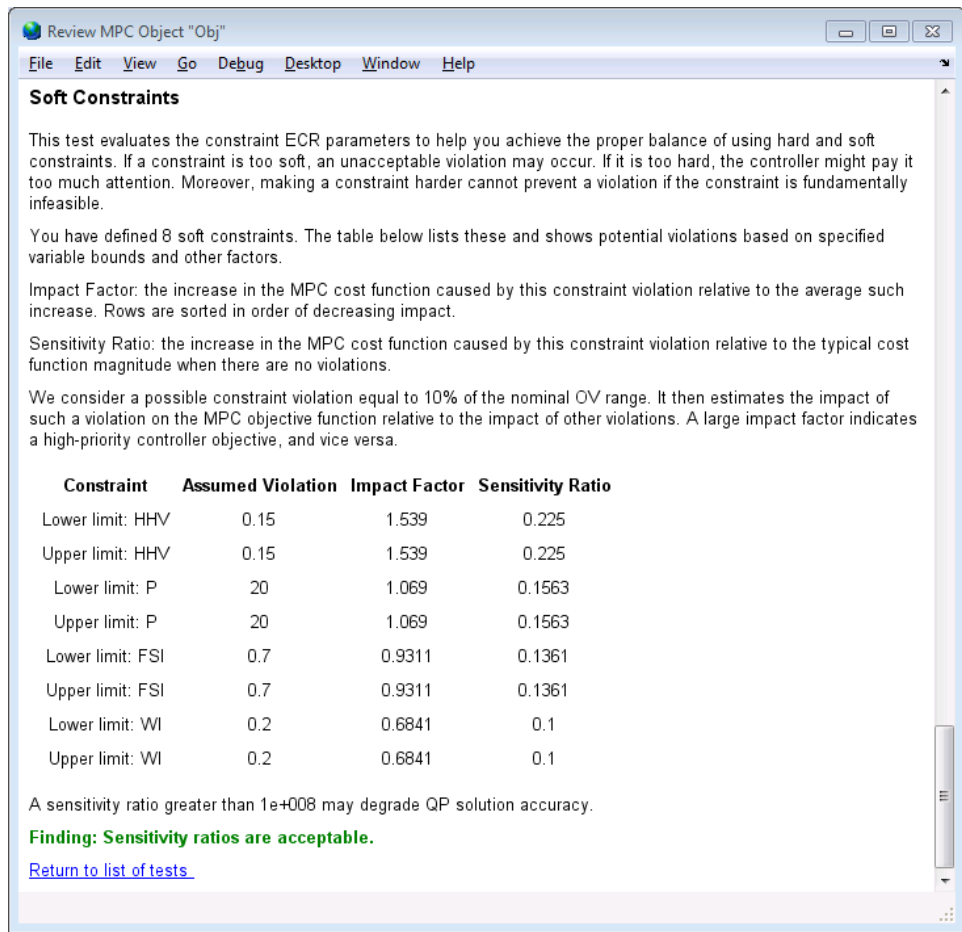
```
Obj.OV(1).MinECR = 0.5;
Obj.OV(1).MaxECR = 0.5;
Obj.OV(3).MinECR = 3;
Obj.OV(3).MaxECR = 3;
Obj.OV(4).MinECR = 80;
Obj.OV(4).MaxECR = 80;
```

Review the impact of this design change.

```
review(Obj)
```


4-107

Notice from the Sensitivity Ratio column that all the sensitivity ratios are now less than unity. This means that the soft constraints will receive less attention than other terms in the MPC objective function, such as deviations of the MVs from their target values. Thus, it is likely that an output constraint violation would occur.

In order to give the output constraints higher priority than other MPC objectives, increase the Weights.ECR parameter from default 1e5 to a higher value to harden all the soft OV constraints.

```
Obj.Weights.ECR = 1e8;
```

Review the impact of this design change.

```
review(Obj)
```

The controller is now a factor of 100 more sensitive to output constraint violations than to errors in target tracking.

```
mpcverbosity(MPC_verbosity);
[~, hWebBrowser] = web;
close(hWebBrowser);
```

Bibliography

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

[2] Rawlings, J. B., and David Q. Mayne "Model Predictive Control: Theory and Design" Nob Hill Publishing, 2010.

Reference for the Design Tool GUI

This chapter is the reference manual for the Model Predictive Control Toolbox design tool (graphical user interface).

- "Working with the Design Tool" on page 5-2
- "Analyze Sensitivity Using the Tuning Advisor" on page 5-63
- "Customize Response Plots" on page 5-71

Working with the Design Tool

In this section...

Opening the MPC Design Tool

To open the Design Tool in MATLAB, type

```
mpctool
```

The design tool is part of the Control and Estimation Tools Manager. When invoked as shown above, the design tool opens and creates a new *project* named **MPC Design Task**.

If you started the tool previously, the above command makes the tool visible but does not create a new project.

Alternatively, if your Simulink model contains a Model Predictive Controller block, you can double-click the block to obtain its mask (see example below)

and click the **Design** button. If the **MPC controller** field is empty, the design tool will create a default controller. Otherwise, it will load the named controller object, which must be in your base workspace. You can then view and modify the controller design.
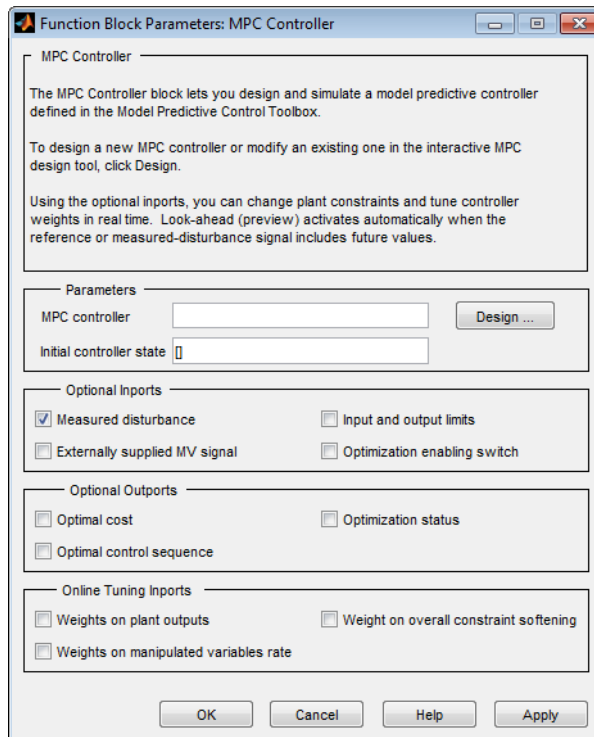
Creating a New MPC Design Task

To create a new **MPC Design Task**:

1 Select the **Workspace** node in the Control and Estimation Tools Manager.

2 Click **File > New > Task**.

3 In the New Project dialog box, click the **Select** check box for **Model Predictive Control Task: MPC Controller** or **Model Predictive Control Task: Multiple MPC Controllers**.

Click **OK**.

Menu Bar

The design tool's menu bar appears whenever you've selected a Model Predictive Control Toolbox project or task in the tree (see "Tree View" on page 5-7). The menu bar's MPC option distinguishes it from other control and estimation tools. See the example below. The following sections describe each menu option.
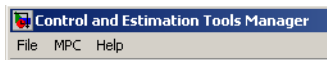
File Menu

- "Load" on page 5-4
- "Save" on page 5-5
- "Close" on page 5-5

Load. Loads a saved design. A dialog box asks you to specify the MAT-file containing the saved design. If the MAT-file contains multiple projects, you must select the one(s) to be loaded (see example below).
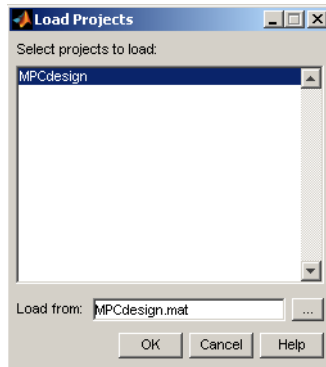
You can also load a design using the toolbar (see "Toolbar" on page 5-6).

Save. Saves a design so you can use it later. The data are saved in a MAT-file. A dialog allows you to specify the file name (see below). If you are working on multiple projects, you can select those to be saved.
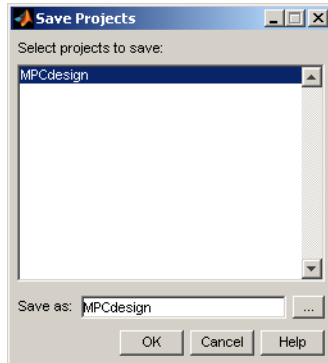
You can also select the **Save** option using the toolbar (see "Toolbar" on page 5-6).

Close. Closes the design tool. If you've modified the design, you'll be asked whether or not you want to save it before closing.

MPC Menu

- "Import" on page 5-5
- "Export" on page 5-6
- "Simulate" on page 5-6

Import. You have the following options:

- **Plant model** – Import a plant model using the model import dialog box (see "Importing a Plant Model" on page 5-8).

- **Controller** – Import a controller using the controller import dialog box (see "Importing a Controller" on page 5-14).

Export. Export a controller using the export dialog box (see "Exporting a Controller" on page 5-16). This option is disabled until your project includes at least one controller.

Simulate. Simulate the *current scenario*, i.e., the one most recently simulated or selected in the tree (see "Tree View" on page 5-7). You can select this option from the keyboard by pressing **Ctrl+R**, or using the toolbar icon (see "Toolbar" on page 5-6).

The **Simulate** option is disabled until your project includes at least one valid simulation scenario.

Toolbar

The toolbar, shown in the following figure, lets you perform the following tasks:

- Load a saved design
- Simulate a current scenario
- Save the current design
- Toggle the output area

For more information on the first three tasks, see the following:

- "Load" on page 5-4
- "Save" on page 5-5
- "Simulate" on page 5-6

The *text output area* is at the bottom of the tool. It displays progress messages and diagnostics. In the above view, the *toggle* button is pushed in, so the text display area appears. If you are working on a small screen, you might use the toggle button to hide the text area, allowing more room to display the controller design.

Tree View

The tree view appears in a frame on the design tool's left-hand side (see example below). When you select one of the tree's *nodes* (by clicking its name or icon) the larger frame to its right shows a dialog pane that allows you to view and edit the specifications associated with that item.

Node Types
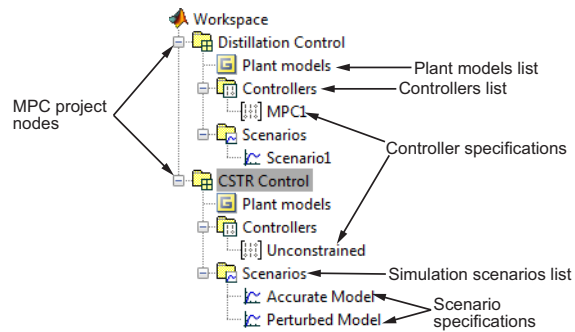
The above example shows two Model Predictive Control Toolbox design project nodes, **Distillation Control** and **CSTR Control**, and their subnodes. For more details on each node type, see the following:

- MPC design project/task – See "Signal Definition View" on page 5-18.

- Plant models list – See "Plant Models View" on page 5-23.

- Controllers list – See "Controllers View" on page 5-25.

- Controller specifications – See "Simulation Scenarios List" on page 5-29.

- Scenarios list – See "Simulation Scenario View" on page 5-54.

- Scenario specifications – See "Controller Specifications View" on page 5-32.

Renaming a Node

You can rename following node types:

- MPC design project/task

- Controller specifications

- Scenario specifications

To rename a node, do *one* of the following:

- Click the name, wait for an edit box to appear, type the desired name, and press the **Enter** key to finalize your choice.

- Right-click the name, select the **Rename** menu option, and enter the desired name in the dialog box.

- To rename a controller, select **Controllers** and edit the controller name in the table.

- To rename a scenario, select **Scenarios** and edit the scenario name in the table.

Importing a Plant Model

To import a plant model, do *one* of the following:

- Select the **MPC/Import/Plant Model** menu option.

- Select the **MPC project/task** node in the tree (see "Tree View" on page 5-7), and then click the **Import Plant** button.

- Right-click the **MPC project/task** node and select the **Import Plant Model** menu option.

- If you've already imported a model, select the **Plant models** node, and then click the **Import** button, or right-click the **Plant models** node and select the **Import Model** menu option.

The **Plant Model Importer** dialog box opens (see the dialog box in "Import from" on page 5-14 for an example). Within the dialog box you can import an LTI model from the workspace or, when you have Simulink Control Design software, you can import a linearized plant model from the Simulink model.

> **Note** Once you have imported a model, any additional models you import to the same MPC project or task must have the identical structure, i.e., the same number of input and output signals, each appearing in the same sequence and having the same signal type designations. If you attempt to import a model that violates one of these conditions, the design tool issues a warning message. If you persist, all previously loaded models will be deleted and the controller design will be re-initialized using the latest model.

The following sections describe model import options:

- "Import from" on page 5-14
- "Import to" on page 5-16
- "Buttons" on page 5-16
- "Importing a Linearized Plant Model" on page 5-11

Import from

Use these options to set the location from which the model will be imported.

MATLAB workspace.
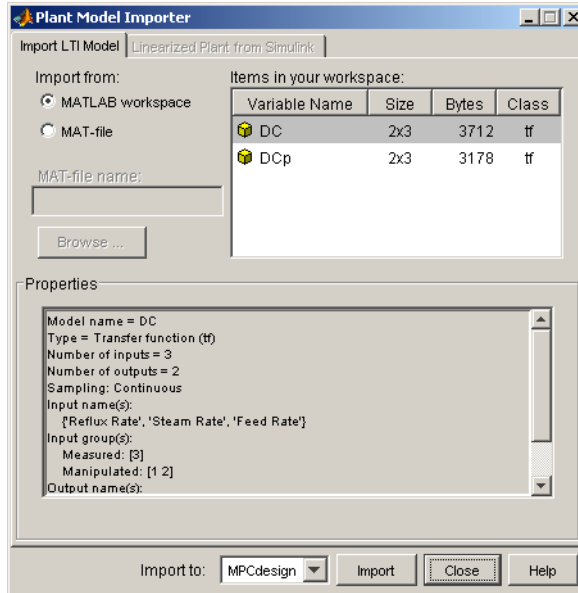
This is the default option and is the case shown in the above example. The **Items in your workspace** area in the upper-right corner lists all candidate models in your MATLAB workspace. Select one by clicking it. The **Properties** area lists the selected model's properties (the DC model in the above example).

MAT-file. The upper part of the dialog box changes as shown below.
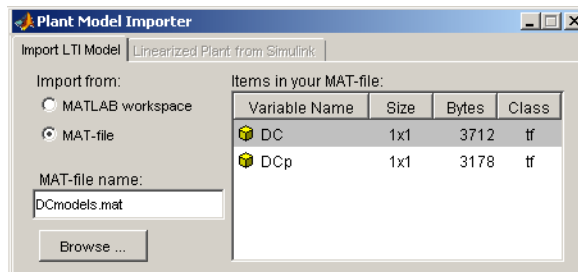
The **MAT-file name** edit box becomes active. Type the desired MAT-file
name (if it's not on the MATLABpath, enter the complete file path). You can
also use the **Browse** button which opens a file chooser window.

In the above example, file `DCmodels.mat` contains two models. Their names
appear in the **Items in your MAT-file** area in the upper-right corner. As
with the workspace option, the selected model's properties appear in the
Properties area.

Import to

The combo box at the bottom of the dialog box allows you to specify the MPC
project/task into which the plant model will be imported (see example below).
It defaults to the active project.

Buttons

Import. Select the model you want to import from the **Items** list in the
upper-right corner of the dialog box. Verify that the **Import to** option
designates the correct project/task. Click the **Import** button to import the
model.

To verify that the model has been loaded, select **Plant models** in the tree.
(See "Tree View" on page 5-7, and "Plant Models View" on page 5-23.)

The import dialog box remains visible until you close it so you can import
additional models.

Close. Click **Close** to close the dialog box. You can also click the Close icon
on the title bar.

Importing a Linearized Plant Model

1 Open the design tool from within a Simulink model as discussed in
"Opening the MPC Design Tool" on page 5-2.

2 Open the **Plant Model Importer** dialog box (see "Importing a Plant Model" on page 5-8).

3 Click the **Linearized Plant from Simulink** tab (see the following example).

Note If you haven't activated the design tool within Simulink, the **Linearized Plant from Simulink** tab is unavailable.
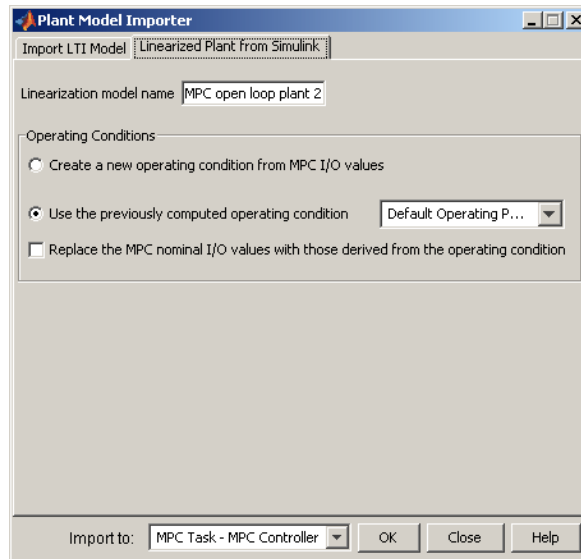
Linearization Process. If you click **OK**, the design tool uses Simulink Control Design software to create a linearized plant model. It performs the following tasks automatically:

1 Configures the Control and Estimation Tools Manager.

2 Temporarily inserts linearization input and output points in the Simulink model at the inputs and outputs of the MPC block.

3 When the **Create a new operating condition from MPC I/O values** option is selected, the Model Predictive Control Toolbox software

temporarily inserts output constraints at the inputs/outputs of the MPC block.

4 Finds a steady state operating condition based on the constraints or uses the specified operating condition.

5 Linearizes the plant model about the operating point.

The linearized plant model appears as a new node under **Plant Models**. For details of the linearization process, refer to the Simulink Control Design documentation.

Linearization Options. You can customize the linearization process in several ways:

- To specify a name for the linearized plant model, enter the name in the **Linearization model name** edit box.

- To use an alternative operating condition, you can:
  - Select one from the menu next to **Use the previously computed operating condition**. This list contains all operating conditions that exist within the current project.
  - Select **Create a new operating condition from MPC I/O values** to compute an operating condition by optimization, using the nominal plant values as constraints. See "Linearize Simulink Models" for an example involving a nonlinear chemical reactor.

- To replace the nominal plant values with the operating point used in the linearization, select the check box next to **Replace the MPC nominal I/O values with those derived from the operating condition**.

- When there are multiple MPC blocks in the Simulink diagram, use the **Import to** menu to select the one that will receive the plant model.

> **Note** The above linearization process automatically identifies the plant's input and output variables according your signal connections in the Simulink model. The controller block does not allow signals corresponding to unmeasured disturbance or unmeasured output variables. Consequently, such variables cannot be included in a model created via the above linearization procedure. If you must include such variables in your controller design, use the Simulink Control Design tool to designate the signals to be used, linearize the plant, and then import this linearized model into the MPC design tool. See "Linearize Simulink Models" for an example of this procedure.

Importing a Controller

To import a controller, do *one* of the following:

- Select the **MPC/Import/Controller** menu option.

- Select the **MPC project/task** node in the tree (see "Tree View" on page 5-7), and then click the **Import Controller** button.

- Right-click the **MPC project/task** node and select the **Import Controller** menu option.

- If you've already designed a controller, select the **Controllers** node and then click the **Import** button, or right-click the **Controllers** node and select the **Import Controller** menu option.

The MPC Controller Importer dialog box opens. The following sections describe its options:

- "Import from" on page 5-14

- "Import to" on page 5-16

- "Buttons" on page 5-16

Import from

Use these options to set the location from which the controller will be imported.

MATLAB Workspace.
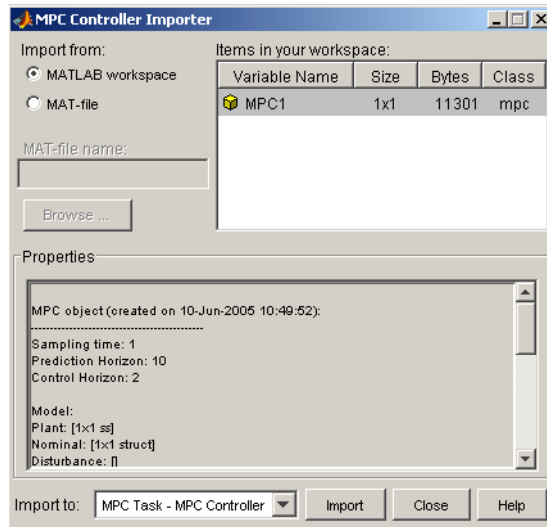
This is the default option and is the case shown in the above example. The **Items in your workspace** area in the upper-right corner lists all MPC objects in your workspace. Select one by clicking it. The **Properties** area lists the properties of the selected model.

MAT-File. The upper part of the dialog box changes as shown below.
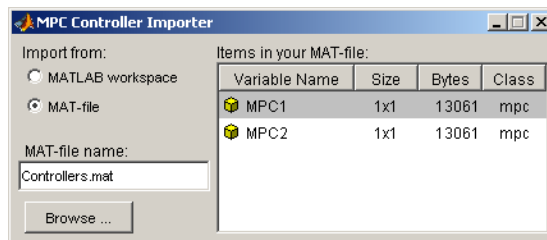
The **MAT-file name** edit box becomes active. Type the desired MAT-file name here (if it's not on the MATLAB path, enter the complete file path). You can also use the **Browse** button which opens a standard file chooser dialog box.

In the above example, file `Controllers.mat` contains two MPC objects. Their names appear in the **Items in your MAT-file** area in the upper-right corner.

Import to

This allows you to specify the MPC task into which the controller will be imported (see example below). It defaults to that most recently active.

Buttons

Import. Select the controller you want to import from the **Items** list in the upper-right corner. Verify that the **Import to** option designates the correct project/task. Click the **Import** button to import the controller.

The new controller should appear in the tree as a subnode of **Controllers**. (See "Tree View" on page 5-7.)

The imported controller contains a plant model, which appears in the **Plant models** list. (See "Plant Models View" on page 5-23.)

Note If the selected controller is incompatible with any others in the designated project, the design tool will not import it.

Close. Click **Close** to close the dialog box. You can also click the Close icon on the title bar.

Exporting a Controller

To export a controller, do *one* of the following:

- Select the **MPC/Export** menu option.
- Select **Controllers** in the tree and click its **Export** button.
- In the tree, right-click **Controllers** and select the **Export Controller** menu option.

- In the tree, right-click the controller you want to export and select the **Export Controller** menu option.

The MPC Controller Exporter dialog box opens (see example below). The following sections describe its options:

Dialog Box Options
The following sections describe the dialog box options.

Controller source. Use this to select the project/task containing the controller to be exported. It defaults to the project/task most recently active.

Controller to export. Use this to specify the controller to be exported. It defaults to the controller most recently selected in the tree.

Name to assign. Use this to assign a valid MATLAB variable name (no spaces). It defaults to the selected controller's name (with spaces removed, if any).

Export to MATLAB workspace. Select this option if you want the controller to be exported to the MATLAB workspace.

Export to MAT-file. Select this option if you want the controller to be exported to a MAT-file.

Buttons

Export. If you've selected the **Export to MATLAB workspace** option, clicking **Export** causes a new MPC object to be created in your MATLAB workspace. (If one having the assigned name already exists, you'll be asked if you want to overwrite it.) You can use the MATLAB `whos` command to verify that the controller has been exported.

If you've selected the **Export to MAT-file** option, clicking **Export** opens a standard file chooser that allows you to specify the file.

In either case, the dialog box remains visible, allowing you to export additional controllers.

Close. Click **Close** to close the dialog box. You can also click the Close icon on the title bar.

Signal Definition View

The signal definition view appears whenever you select a Model Predictive Control Toolbox project or task node in the tree (see "Tree View" on page 5-7). You'll see this view when you open the design tool for the first time. An example appears below.
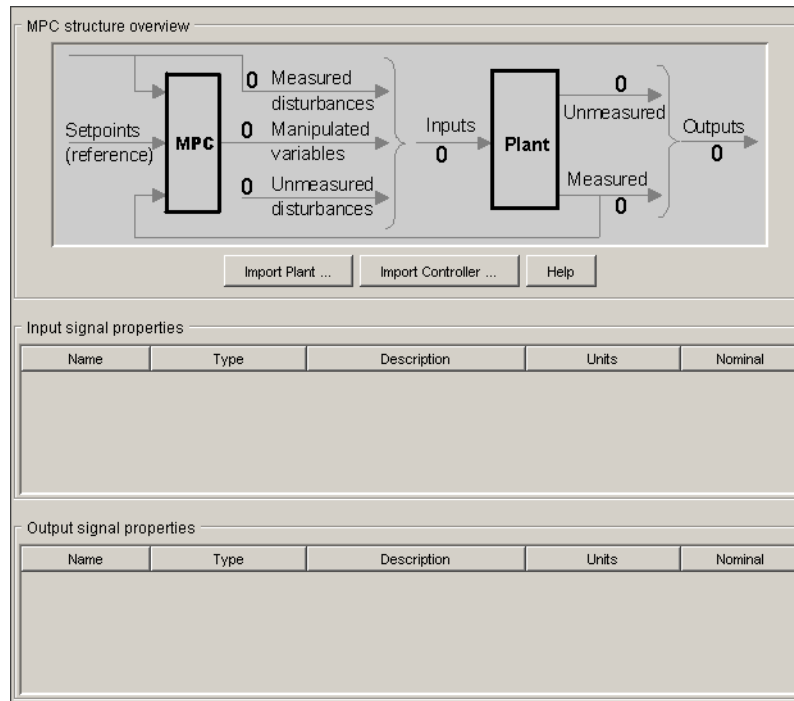
The following sections describe the view's main features:

MPC Structure Overview

This upper section is a noneditable display of your application's structure. Once you've imported a plant model (or controller), tool counts and displays the five possible signal types, as in the example below.
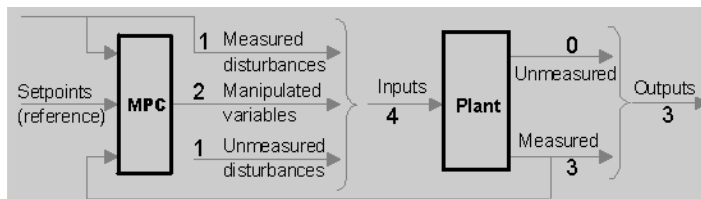
The counts change if you edit the signal types.

Buttons

Import Plant. Clicking this opens the Plant Model Importer dialog box (see "Importing a Plant Model" on page 5-8).

Import Controller. Clicking this opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 5-14).

Note You won't be allowed to proceed with your design until you import a plant model. You can do so indirectly by importing a controller or loading a saved project.

Signal Properties Tables
Two tables display the properties of each signal in your design.

Input Signal Properties. The plant's input signals appear as table rows (see example below).

The entries are editable and have the following significance:

- **Name** – The signal name, an alphanumeric string used to label other tables, graphics, etc. Each name must be unique. The design tool assigns a default name if your imported model doesn't specify one.

- **Type** – One of the three valid Model Predictive Control Toolbox input signal types. The above example shows one of each. To change a signal's type, click the table cell and select the desired type. The options are as follows:

  Manipulated – A signal that will be manipulated by the controller, i.e., an actuator (valve, motor, etc.).

  Measured Disturbance – An independent input whose value is measured and used as a controller input for *feedforward compensation*.

  Unmeasured Disturbance – An independent input representing an unknown, unpredictable disturbance.

- **Description** – An optional descriptive string.

- **Units** – Optional units (dimensions), a string. Used to label other dialog boxes, simulation plots, etc.

- **Nominal** – The signal's nominal value. The design tool defaults this to zero. Any value you assign here will be the default initial condition in simulations.

Note Your design must include at least one *manipulated variable*. The other input signal types need not be included.

Output Signal Properties. The plant's output signals appear as table rows (see example below).

Output signal properties				
Name	Type	Description	Units	Nominal
H_2	Measured	Headbox level	m	0.0
N_1	Measured	Feed tank consistency	%	0.0
N_2	Measured	Head box consistency	%	0.0

The entries are editable and have the following significance:

- **Name** – The signal name, an alphanumeric string used to label other tables, graphics, etc. Each name must be unique. The design tool assigns a default name if your imported model doesn't specify one.

- **Type** – One of the two valid Model Predictive Control Toolbox output signal types. The above example shows one of each. To change a signal's type, click the table cell and select the desired type. The options are as follows:

  Measured – A signal the controller can use for feedback.

  Unmeasured – Predicted by the plant model but unmeasured. Can be used as an indicator. Can also be assigned a setpoint or constrained.

- **Description** – An optional descriptive string.

- **Units** – Optional units (dimensions), a string. Used to label other dialog boxes, simulation plots, etc.

- **Nominal** – The signal's nominal value. The design tool defaults this to zero. Any value you assign here will be the default initial condition in simulations.

Note Your design must include at least one *measured* output. Inclusion of unmeasured outputs is optional.

Right-Click Menu Options

Right-clicking on an MPC project/task node allows you to choose one of the following menu items:

- **Import Plant Model** – Opens the Plant Model Importer dialog box (see "Importing a Plant Model" on page 5-8).

- **Import Controller** – Opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 5-14).

- **Clear Project** – Erases all plant models, controllers, and scenarios in your design, returning the project to its initial empty state.

- **Delete Project** – Deletes the selected project node.

Plant Models View

Selecting **Plant models** in the tree dispays this view (see example below).

The following sections describe the view's main features:

- "Plant Models List" on page 5-24

- "Model Details" on page 5-24

- "Additional Notes" on page 5-25

- "Buttons" on page 5-25

- "Right-Click Options" on page 5-25

Plant Models List

This table lists all the plant models you've imported and/or plant models contained in controllers that you've imported. The example below lists two imported models, DC and DCp .

The **Name** field is editable. Each model must have a unique name. The name you assign here will be used within the design tool only.

The **Type** field is noneditable and indicates the model's LTI object type (see the Control System Toolbox documentation for a detailed discussion of LTI models).

The **Sampling Period** field is zero for continuous-time models, and a positive real value for discrete-time models.

The **Imported on** field gives the date and time the model was imported.

Model Details

This scrollable viewport shows details of the model currently selected in the plant models list (see "Plant Models List" on page 5-24). An example appears below.

Additional Notes

You can use this editable text area to enter comments, distinguishing model features, etc.

Buttons

Import. Opens the Plant Model Importer dialog box (see "Importing a Plant Model" on page 5-8).

Delete. Deletes the selected model. If the model is being used elsewhere (i.e., in a controller or scenario), the first model in the list replaces it and a warning message appears.

Right-Click Options

Right-clicking the **Plant models** node causes the following menu option to appear.

Import Model. Opens the Plant Model Importer dialog box (see "Importing a Plant Model" on page 5-8).

Controllers View

Selecting **Controllers** in the tree displays this view (see example below).
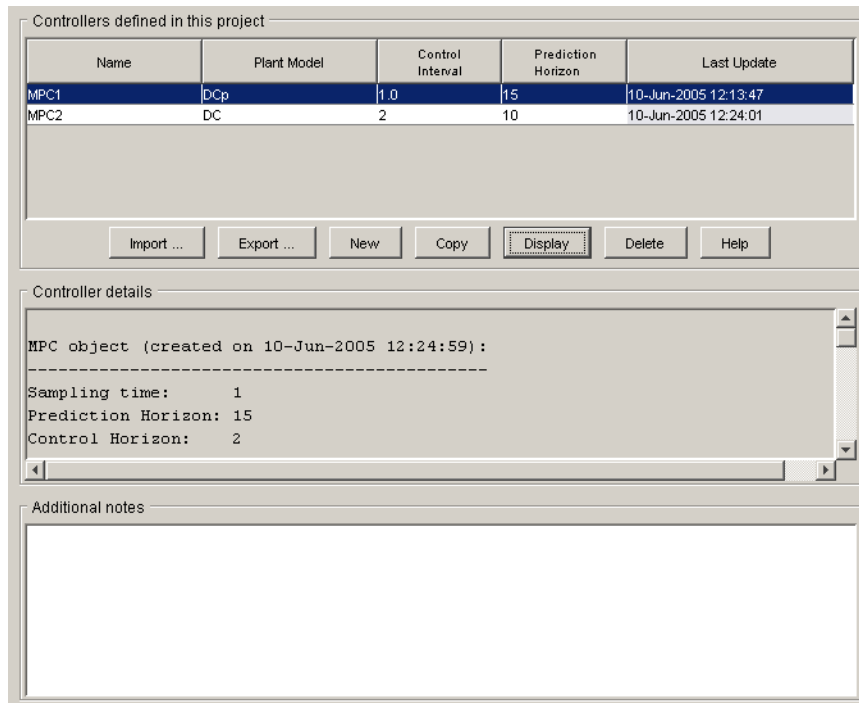
The following sections describe the view's main features:

- "Controllers List" on page 5-26
- "Controller Details" on page 5-27
- "Additional Notes" on page 5-28
- "Buttons" on page 5-28
- "Right-Click Options" on page 5-29

Controllers List

This table lists all the controllers in your project. The example below lists two controllers, MPC1 and MPC2 .

The **Name** field is editable. The name you assign here must be unique. You will refer to it elsewhere in the design tool, e.g., when you use the controller in a simulation scenario. Each listed controller corresponds to a subnode of **Controllers** (see "Tree View" on page 5-7). Editing the name in the table will rename the corresponding subnode.

The **Plant Model** field is editable. To change the selection, click the cell and choose one of your models from the list. (All models appearing in the Plant Models view are valid choices. See "Plant Models View" on page 5-23.)

The **Control Interval** field is editable and must be a positive real number. You can also set it in the Controller Specifications view (see "Model and Horizons Tab" on page 5-33 for more details).

The **Prediction Horizon** field is editable and must be a positive, finite integer. You can also set in the Controller Specifications view (see "Model and Horizons Tab" on page 5-33 for more details).

The noneditable **Last Update** field gives the date and time the controller was most recently modified.

Controller Details

This scrollable viewport shows details of the controller currently selected in the controllers list (see "Controllers List" on page 5-26). An example appears below.
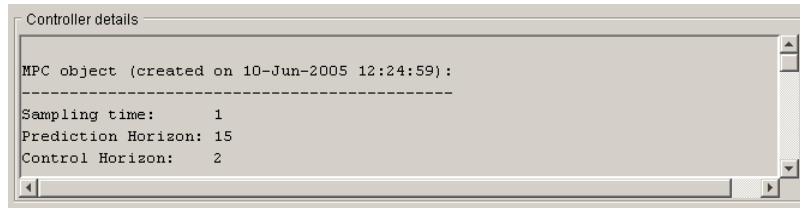
> **Note** This view shows controller details once you have used the controller in a simulation. Prior to that, it is empty. If necessary, you can use the **Display** button to force the details to appear.

Additional Notes

You can use this editable text area to enter comments, distinguishing controller features, etc.

Buttons

Import. Opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 5-14).

Export. Opens the MPC Controller Exporter dialog box (see "Exporting a Controller" on page 5-16).

New. Creates a new controller specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Copy. Copies the selected controller, creating a controller specification subnode containing the same controller settings, and assigning it a default name.

Display. Calculates and displays details for the selected controller.

Delete. Deletes the selected controller. If the controller is being used elsewhere (i.e., in a simulation scenario), the first controller in the list replaces it (and a warning message appears).

Right-Click Options

Right-clicking the **Controllers** node causes the following menu options to appear.

New Controller. Creates a new controller specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Import Controller. Opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 5-14).

Export Controller. Opens the MPC Controller Exporter dialog box (see "Exporting a Controller" on page 5-16).

Simulation Scenarios List

Selecting **Scenarios** in the tree causes this view to appear (see example below).
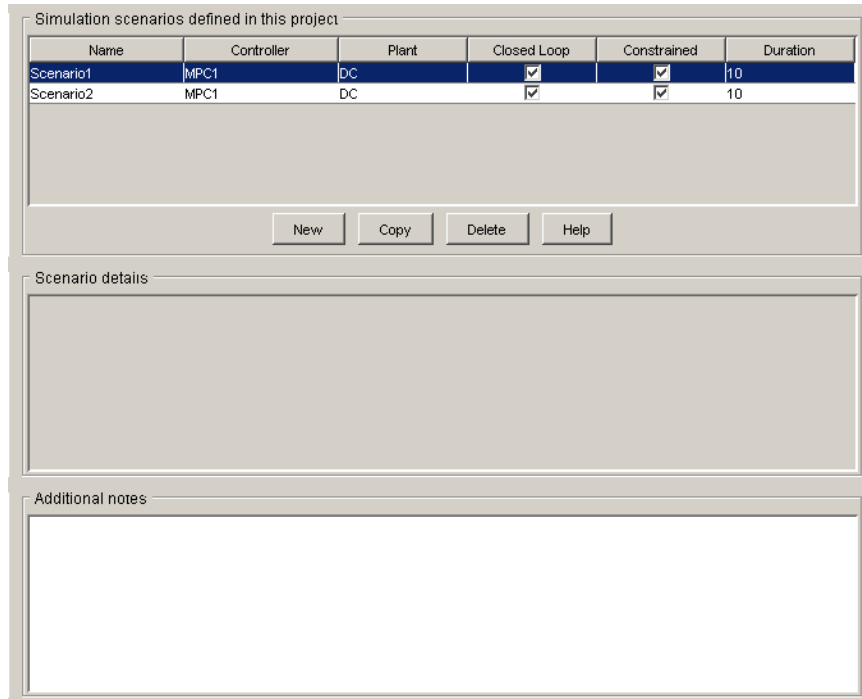
The following sections describe the view's main features:

- "Scenarios List" on page 5-30
- "Scenario Details" on page 5-32
- "Additional Notes" on page 5-32
- "Buttons" on page 5-32
- "Right-Click Options" on page 5-32

Scenarios List

This table lists all the scenarios in your project. The example below lists two, Scenario1 and Scenario2 .

The **Name** field is editable. The assigned name must be unique. Each listed scenario corresponds to a subnode of **Scenarios** (see "Tree View" on page 5-7). Editing the name in the table will rename the corresponding subnode.

The **Controller** field is editable. To change the selection, click the cell and select one of your controllers from the list. (All controllers appearing in the Controllers view are valid choices. See "Controllers View" on page 5-25.) You can also set this using the Scenario Specifications view (for more discussion, see "Simulation Scenario View" on page 5-54).

The **Plant** field is editable. To change the selection, click the cell and select one of your plant models from the list. (All models appearing in the Plant Models view are valid choices. See "Plant Models View" on page 5-23.) You can also set this in the scenario specifications (for more discussion, see "Simulation Scenario View" on page 5-54).

The **Closed Loop** field is an editable check box. If cleared, the simulation will be open loop. You can also set it in the scenario specifications (for more discussion see "Simulation Scenario View" on page 5-54).

The **Constrained** field is an editable check box. If cleared, the simulation will ignore all constraints specified in the controller design. You can also set it in the scenario specifications (for more discussion see "Simulation Scenario View" on page 5-54).

The **Duration** field is editable and must be a positive, finite real number. It sets the simulation duration. You can also set it in the scenario specifications (for more discussion, see "Simulation Scenario View" on page 5-54).

Scenario Details

This area is blank at all times.

Additional Notes

You can use this editable text area to enter comments, distinguishing scenario features, etc.

Buttons

New. Creates a new scenario specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Copy. Copies the selected scenario, creating a scenario specification subnode containing the same settings, and assigning it a default name.

Delete. Deletes the selected scenario.

Right-Click Options

Right-clicking the **Scenarios** node causes the following menu option to appear

New Scenario. Creates a new scenario specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Controller Specifications View

This view appears whenever you select one of your controller nodes (see "Tree View" on page 5-7). It allows you to review and edit controller settings. It consists of four tabs, each devoted to a particular design aspect. All settings have default values.

The following sections describe the view's main features:

- "Model and Horizons Tab" on page 5-33
- "Constraints Tab" on page 5-36
- "Constraint Softening" on page 5-38

- "Weight Tuning Tab" on page 5-42

- "Estimation Tab" on page 5-45

- "Right-Click Menus" on page 5-54

Model and Horizons Tab

Plant Model.

This combo box allows you to specify the plant model the controller uses for its predictions. You can choose any of the plant models you've imported. (See "Importing a Plant Model" on page 5-8.)

Horizons.

The **Control interval** option sets the elapsed time between successive controller moves. It must be a positive, finite real number. The calculations assume a zero-order hold on the manipulated variables (the signals adjusted by the controller). Thus, these signals are constant between moves.

The **Prediction horizon** option sets the number of *control intervals* over which the controller predicts its outputs when computing controller moves. It must be a positive, finite integer.

The **Control horizon** option sets the number of moves computed. It must be a positive, finite integer, and must not exceed the prediction horizon. If less than the prediction horizon, the final computed move fills the remainder of the prediction horizon.
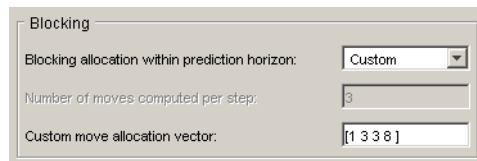
For more discussion, see "Typical Sampling Instant" on page 1-14, and "Prediction and Control Horizons" on page 1-18.

Blocking.

By default, the **Blocking** option is cleared (off). When selected as shown above, the design tool replaces the **Control horizon** specification (see "Horizons" on page 5-34) with a move pattern determined by the following settings:

- **Blocking allocation within prediction horizon** – Choices are:

  Beginning – Successive moves at the beginning of the prediction horizon, each with a duration of one control interval.

  Uniform – The prediction horizon is divided by the number of moves and rounded to obtain an integer duration, and each computed move has this duration (the last move extends to fill the prediction horizon).

  End – Successive moves at the end of the prediction horizon, each with a duration of one control interval.

  Custom – You specify the duration of each computed move.

- **Number of moves computed per step** – The number of moves computed when the allocation setting is **Beginning**, **Uniform**, or **End**. Must be a positive integer not exceeding the prediction horizon.

- **Custom move allocation vector** – The duration of each computed move, specified as a row vector. In the example below, there are four moves, the first lasting 1 control interval, the next two lasting 3, and the final lasting 8 for a total of 15. The **Number of moves computed per step** setting is disabled (ignored).

The sum of the vector elements should equal the prediction horizon (15 in this case). If not, the last move is extended or truncated automatically.

Note When **Blocking** is off, the controller uses the **Beginning** allocation with **Number of moves computed per step** equal to the **Control horizon**.

For more discussion, see "Blocking" on page 1-9.

Constraints Tab

This tab allows you to specify *constraints* (bounds) on *manipulated variables* and *outputs*. Constraints can be *hard* or *soft*. By default, all variables are *unconstrained*, as shown in the view below.

| Model and Horizons | Constraints | Weight Tuning | Estimation (Advanced) |

Constraints on manipulated variables

Name	Units	Minimum	Maximum	Max Down Rate	Max Up Rate
Reflux Rate					
Steam Rate					

Constraints on output variables

Name	Units	Minimum	Maximum
Distillate Purity			
Bottoms Purity			

Constraint Softening   Help

Note If you specify constraints, manipulated variable constraints are hard by default, whereas output variable constraints are soft by default. You can customize this behavior, as discussed in the following sections. For additional information on constraints, see "Optimization and Constraints" on page 1-5, and "Optimization Problem" on page 2-5.

Each table entry may be a *scalar* or a *vector*. A scalar entry defines a constraint that is constant for the entire prediction horizon. A vector entry defines a time-varying constraint. See "Entering Vectors in Table Cells" on page 5-44 for the required format.

An entry may also be any valid MATLAB expression provided that it evaluates to yield an appropriate scalar or vector quantity.

Constraints on Manipulated Variables. The example below is for an application with two manipulated variables (MVs), each represented by a table row.

Name	Units	Minimum	Maximum	Max Down Rate	Max Up Rate
Reflux Rate		0	85	-10	10
Steam Rate		0	52		

Constraints on manipulated variables

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The remaining entries are editable. If you leave a cell blank, the controller ignores that constraint. You can achieve the same effect by entering `-Inf` or `Inf` (for a minimum or maximum, respectively).

The **Minimum** and **Maximum** values set each MV's range.

The **Max down rate** and **Max up rate values** set the amount the MV can change *in a single control interval*. The **Max down rate** must be negative or zero. The **Max up rate** must be positive or zero.

Constraint values must be consistent with your nominal values (see "Input Signal Properties" on page 5-20). In other words, each MV's nominal value must satisfy the constraints.

Constraint values must also be self-consistent. For example, an MV's lower bound must not exceed its upper bound.

Constraints on Output Variables. The example below is for an application with two output variables, each represented by a table row.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The remaining entries are editable. If you leave a cell blank (as above), the controller ignores that constraint. You can achieve the same effect by entering -Inf (for a **Minimum**) or Inf (for a **Maximum**).

Constraint values must be consistent with your nominal values (see "Output Signal Properties" on page 5-21). In other words, each output's nominal value must satisfy the constraints.

Constraint values must also be self-consistent. For example, an output's lower bound must not exceed its upper bound.

Note Don't constrain outputs unless this is an essential aspect of your application. It is usually better to define output setpoints (reference values) rather than constraints.

Constraint Softening

A *hard* constraint cannot be violated. Hard constraints are risky, especially for outputs, because the controller will ignore its other objectives in order to satisfy them. Also, the constraints might be impossible to satisfy in certain situations, in which case the calculations are mathematically *infeasible*.

Model Predictive Control Toolbox software allows you to specify *soft* constraints. These can be violated, but you specify a violation tolerance for each (the *relaxation band*). See the example specifications below.

To open this dialog box, click the **Constraint softening** button at the bottom of the **Constraints** tab in the Controller Specification view (see "Constraints Tab" on page 5-36).

As for the constraints themselves, an entry can be a scalar or a vector. The latter defines a time-varying relaxation band. See "Entering Vectors in Table Cells" on page 5-44 for the required format.

Input Constraints. An example input constraint softening specification appears below.

| Input constraints									
Name	Units	Minimum	Min Band	Maximum	Max Band	Max Down...	Max Down...	Max Up Rate	Max Up Ba...
Reflux Rate		0		85		-10		10	
Steam Rate		0	2	52	2				

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Minimum**, **Maximum**, **Max down rate**, and **Max up rate** columns are editable. Their values are the same as on the main **Constraints** tab (see "Constraints on Manipulated Variables" on page 5-37). You can specify them in either location.

The remaining columns specify the *relaxation band* for each constraint. An empty cell is equivalent to a zero, i.e., a hard constraint.

Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

The example above shows a relaxation band of 2 moles/min for the steam flow rate's lower and upper bounds. The lack of a relaxation band setting for the reflux flow rate's constraints means that these will be hard.

Note The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band.

Output Constraints. An example output constraint specification appears below.

Output constraints					
Name	Units	Minimum	Min Band	Maximum	Max Band
Distillate Purity		90	0.5		
Bottoms Purity		93	2		

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Minimum** and **Maximum** columns are editable. Their values are the same as on the main **Constraints** tab (see "Constraints on Output Variables" on page 5-37). You can specify them in either location.

The remaining columns specify the *relaxation band* for each constraint. An empty cell is equivalent to 1.0, i.e., a *soft* constraint.

Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

The example above shows a relaxation band of 0.5 mole % for the distillate purity lower bound, and a relaxation band of 2 mole % for the bottoms purity lower bound (the softer of the two constraints).

Note The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band.

Overall Constraint Softness. The relaxation band settings allow you to adjust the hardness/softness of each constraint. You can also soften/harden all constraints simultaneously using the slider at the bottom of the dialog box pane.
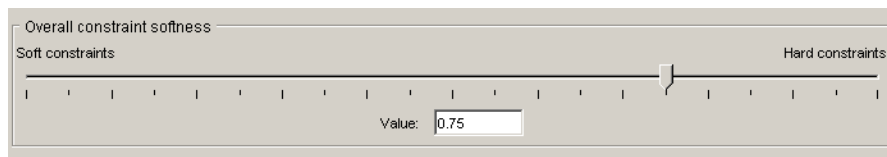
You can move the slider or edit the value in the edit box, which must be between 0 and 1.

Buttons. **OK** – Closes the constraint softening dialog box, implementing changes to the tabular entries or the slider setting.

Cancel – Closes the constraint softening dialog box without changing anything.

Weight Tuning Tab

The example below shows the Model Predictive Control Toolbox default tuning weights for an application with two manipulated variables and two outputs.

The following sections discuss the three tab areas in more detail. For additional information, see "Optimization Problem" on page 2-5.

Each table entry may be a *scalar* or a *vector*. A scalar entry defines a weight that is constant for the entire prediction horizon. A vector entry defines a time-varying weight. See "Entering Vectors in Table Cells" on page 5-44 for the required format.

Input Weights.

Input weights				
Name	Description	Units	Weight	Rate Weight
Reflux Rate	Molar reflux rate	kmol/min	0	0.1
Steam Rate	Steam heating rate	kmol/min	0	0.1

The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Weight** column sets a penalty on deviations of each manipulated variable (MV) from its *nominal value*. The weight must be zero or a positive real number. The default is zero, meaning that the corresponding MV can vary freely provided that it satisfies its constraints (see "Constraints on Manipulated Variables" on page 5-37).

A large **Weight** discourages the corresponding MV from moving away from its nominal value. This can cause *steady state error* (offset) in the output variables unless you have extra MVs at your disposal.

Note To set the nominal values, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Rate Weight** value sets a penalty on MV *changes*, i.e., on the magnitude of each MV move. Increasing the penalty on a particular MV causes the controller to change it more slowly. The table entries must be zero or positive real numbers. These values have no effect in steady state.

Output Weights.

Output weights			
Name	Description	Units	Weight
Distillate Purity	Distillate product purity	mol %	1.0
Bottoms Purity	Bottoms product purity	mol %	1.0

The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Weight** column sets a penalty on deviations of each output variable from its *setpoint* (or *reference*) *value*. The weight must be zero or a positive real number.

A large **Weight** discourages the corresponding output from moving away from its setpoint.

If you don't need to hold a particular output at a setpoint, set its **Weight** to zero. This may be the case, for example, when an output doesn't have a target value and is being used as an indicator variable only.

Overall (Slider Control).
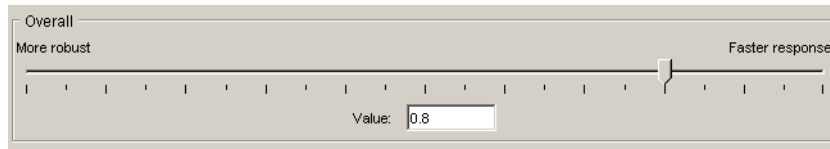
The slider adjusts the weights on all variables simultaneously. Moving the slider to the left increases rate penalties relative to setpoint penalties, which often (but not always!) increases controller robustness. The disadvantage is that disturbance rejection and setpoint tracking become more sluggish.

You can also change the value in the edit box. It must be a real number between 0 and 1. The actual effect is nonlinear. You will generally need to run trials to determine the best setting.

Entering Vectors in Table Cells

In the above examples all constraints and weights were entered as scalars. A scalar entry defines a value that is constant for the entire prediction horizon.

You can also define a constraint or weight that varies with time by entering a *vector*. For the rationale and theoretical basis, see "View and Alter Controller Properties" and "Optimization Problem" on page 2-5.

Enter vectors using the standard MATLAB syntax. For example, `[1, 2, 3]` defines a vector containing three elements, the values 1, 2, and 3.

Entries can be either row or column vectors. A MATLAB expression that produces a vector works too. For example, `4*ones(3,1)` would be a valid entry.

If a vector contains fewer than P elements, where P is the horizon length, the controller automatically extends the vector using its last element. For example, if you entered `[1, 2, 3]` and $P = 5$, the vector used in controller calculations would be `[1, 2, 3, 3, 3]`.

Estimation Tab

Use these specifications to shape the controller's response to unmeasured disturbances and measurement noise.

The example below shows Model Predictive Control Toolbox default settings for an application with two output variables and no unmeasured disturbance inputs.
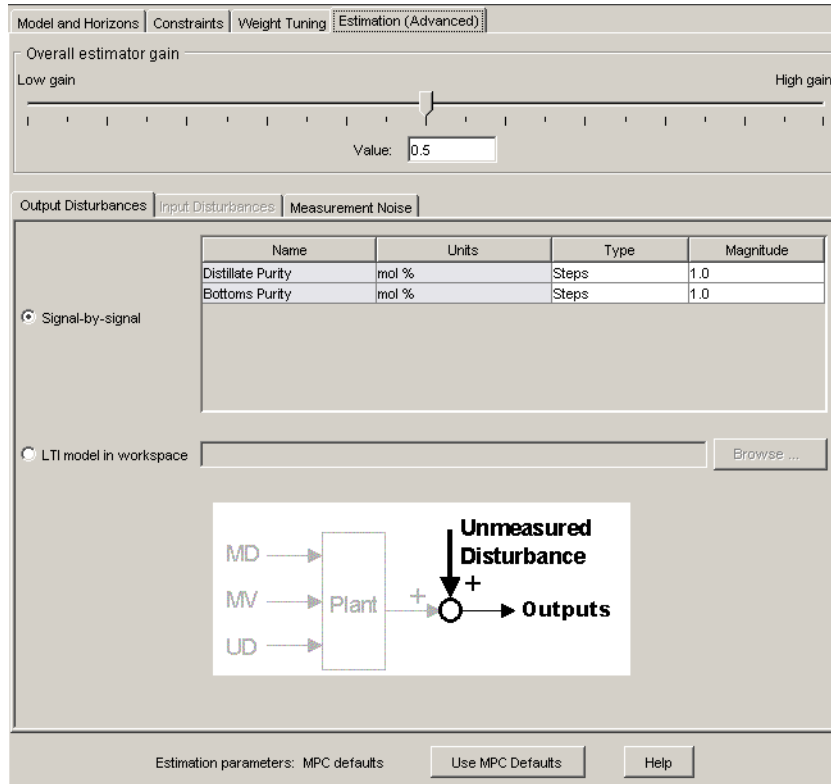
The following sections cover each estimation feature in detail. For additional information, see "Estimating States from Measured Data" on page 1-9 for an introduction, and "State Estimation" on page 2-17 for detailed information.

Button (MPC Default Settings). If you edit any of the **Estimation** tab settings, the display near the top will appear as follows.

To return the settings to the default state, click the **Use MPC Defaults** button, causing the display to revert to the default condition shown below.

Overall Estimator Gain.
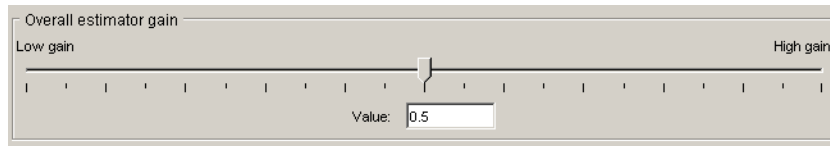
This slider determines the controller's overall disturbance response. As you move the slider to the left, the controller responds less aggressively to unexpected changes in the outputs, i.e., it assumes that such changes are more likely to be caused by measurement noise rather than a *real* disturbance.

You can also change the value in the edit box. It must be between zero and 1. The effect is nonlinear, and you might need to run trial simulations to achieve the desired result.

Output Disturbances

Use these settings to model unmeasured disturbances adding to the plant outputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for an application having two plant outputs.
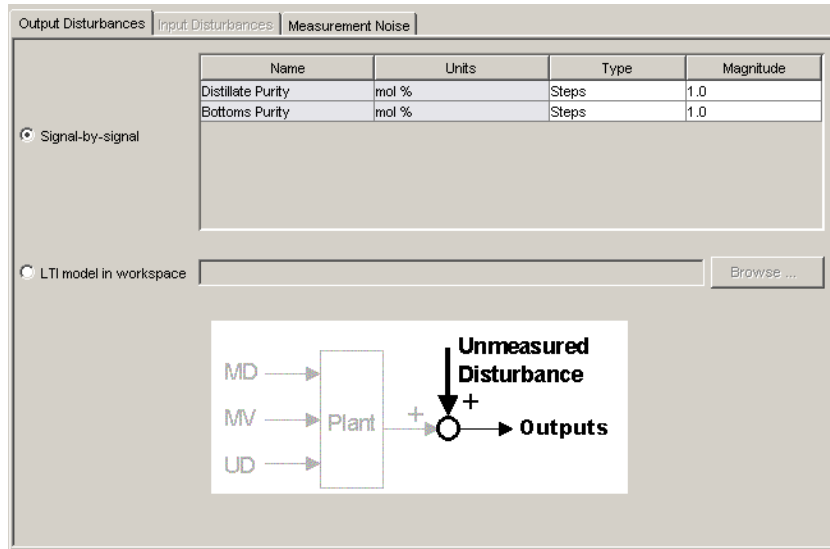
The graphic shows the disturbance location.

Use the table to specify the disturbance character for each output.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **Steps** – Simulates random step-like disturbances (integrated white noise).

- **Ramps** – Simulates a random drifting disturbance (doubly-integrated white noise).

- **White** – White noise.

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the disturbance. Set it to zero if you want to turn off a particular disturbance.

For example, if **Type** is **Steps** and **Magnitude** is 2, the disturbance model is integrated white noise, where the white noise has a standard deviation of 2.

If these options are too restrictive, select the **LTI model in workspace** option. The tab appearance changes to the view shown below.
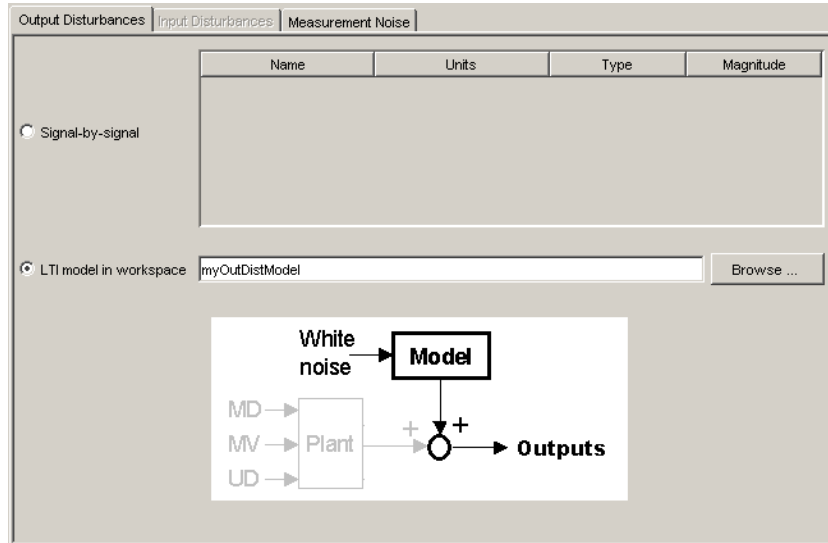
You must specify an LTI output disturbance model residing in your workspace. The **Browse** button opens a dialog box listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The model must have the same number of outputs as the plant.

The white noise entering the model is assumed to have unity standard deviation.

Input Disturbances. Use these settings to model disturbances affecting the plant's unmeasured disturbance inputs.

> **Note** This option is available only if your plant model includes unmeasured disturbance inputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for a plant having one unmeasured disturbance input. The graphic shows the disturbance location.
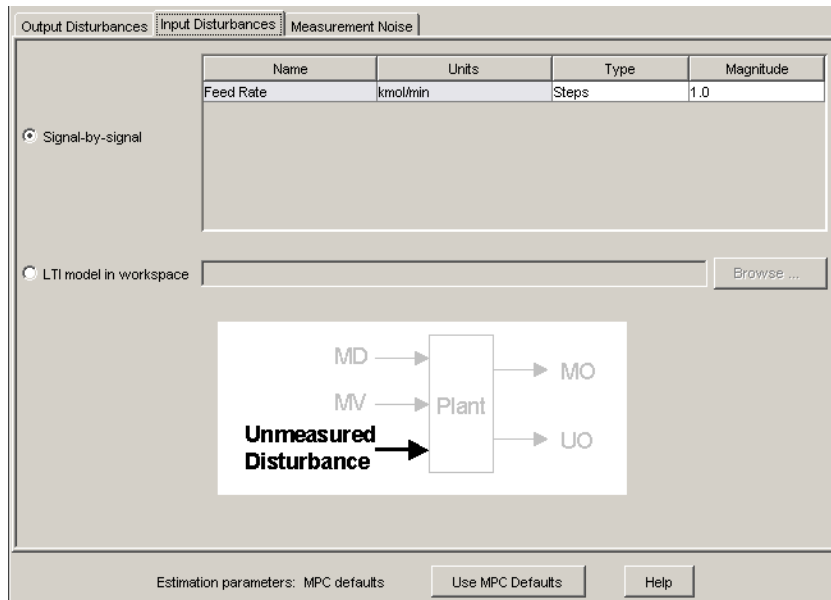
Use the table to specify the character of each unmeasured disturbance input.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

• **Steps** – Simulates random step-like disturbances (integrated white noise).

- **Ramps** – Simulates a random drifting disturbance (doubly-integrated white noise).

- **White** – White noise.

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the disturbance. Set it to zero if you want to turn off a particular disturbance.

For example, if **Type** is **Steps** and **Magnitude** is 2, the disturbance model is integrated white noise, where the white noise has a standard deviation of 2.

If the above options are too restrictive, select the **LTI model in workspace** option. The tab appearance changes to the view shown below.

You must specify an LTI disturbance model residing in your workspace. The **Browse** button opens a dialog box listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The number of model outputs must equal the number of plant unmeasured disturbance inputs. The white noise entering the model is assumed to have unity standard deviation.

Noise. Use these settings to model noise in the plant's measured outputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for a plant having two measured outputs. The graphic shows the noise location.

Use the table to specify the character of each noise input.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes there apply to the entire design.)

The **Type** column sets the noise character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **White** – White noise.

- **Steps** – Simulates random step-like disturbances (integrated white noise).

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the noise. Set it to zero if you want to specify that an output is noise-free.

For example, if **Type** is **Steps** and **Magnitude** is 2, the noise model is integrated white noise, where the white noise has a standard deviation of 2.

If the above options are too restrictive, select the **LTI model in workspace** option. The tab appearance changes as follows.
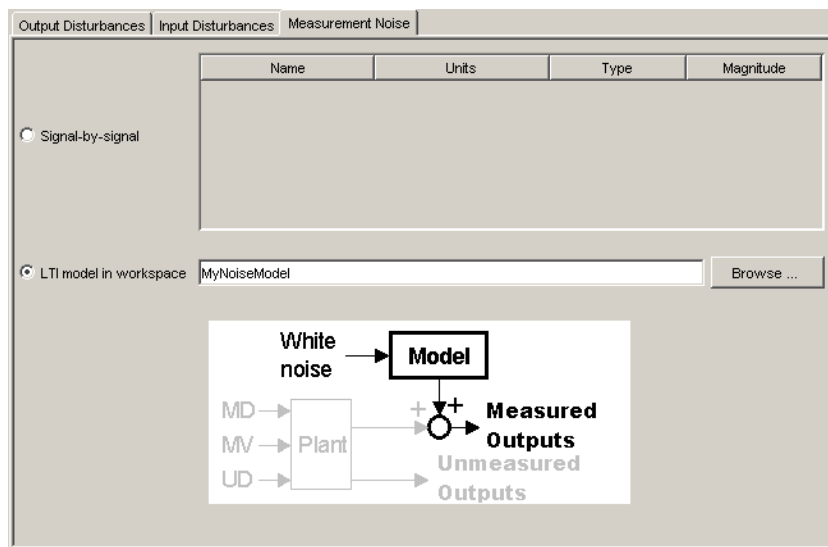
You must specify an LTI model residing in your workspace. The **Browse** button opens a dialog box listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The number of noise model outputs must equal the number of plant measured outputs.

The white noise entering the model is assumed to have unity standard deviation.

Right-Click Menus

Copy Controller. Creates a new controller having the same settings and a default name.

Delete Controller. Deletes the controller. If the controller is being used in a simulation scenario, the design tool replaces it with the first controller in your list, and displays a warning message.

Rename Controller. Opens a dialog box allowing you to rename the controller.

Note Each controller in a design project/task must have a unique name.

Export Controller. Opens the MPC Controller Exporter dialog box (see "Exporting a Controller" on page 5-16).

Simulation Scenario View

This view appears whenever you select one of your scenario specification nodes (see "Tree View" on page 5-7). It allows you to specify simulation settings and independent variables. All have default values, but you will want to change at least some of them (otherwise all independent variables will be constant). Defaults for a plant with three inputs and two outputs appears below.

The middle table won't appear unless you have designated at least one input signal to be a measured disturbance.

The following sections describe the view's main features:

• "Tuning Advisor Button" on page 5-61

• "Right-Click Menus" on page 5-61

Simulation Settings

Use this section to set the following:

• **Controller** – Select one of your controllers,

• **Plant** – Select the plant model that will act as the "real" plant in the simulation, i.e., it need not be the same as that used for controller predictions.

• **Duration** – The simulation duration in time units.

• **Close loops** – If cleared, the simulation will be open-loop.

• **Enforce Constraints** – If cleared, all controller constraints will be ignored.

The **Control interval** field is display-only, and reflects the setting in your **Controller** selection. You can change it there if necessary (see "Model and Horizons Tab" on page 5-33).

Setpoints

Note Setpoint specifications affect *closed-loop* simulations only.

Use this table to specify the setpoint for each output. In the example below, which is for an application having two plant outputs, the first would be constant at 0.0, and the second would change step-wise.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes apply to the entire design.)

The **Type** column specifies the setpoint variation. To change this, click the cell and select a choice from the resulting menu.

The significance of the **Initial Value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-60.

If the **Look Ahead** option is selected (i.e., on), the controller will use future values of the setpoints in its calculations. This improves setpoint tracking, but knowledge of future setpoint changes is unusual in practice.

Note In the current implementation, selecting or clearing the **Look ahead** option for one output will set the others to the same state. Model Predictive Control Toolbox code does not allow you to **Look ahead** for some outputs but not for others.

Measured Disturbances

Use this table to specify the variation of each measured disturbance. In the example below, which is for an application having a single measured disturbance, the "Steam Rate" input would be constant at 0.0.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes apply to the entire design.)

The **Type** column specifies the disturbance variation. To change this, click the cell and select a choice from the resulting menu.

The significance of the **Initial Value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-60.

If the **Look Ahead** option is selected (i.e., on), the controller will use future values of the measured disturbance(s) in its calculations. This improves disturbance rejection, but knowledge of future disturbances is unusual in practice. *It has no effect in an open-loop simulation*.

Note In the current implementation, selecting or clearing the **Look ahead** option for one input will set the others to the same state. Model Predictive Control Toolbox code does not allow you to **Look ahead** for some inputs but not for others.

Unmeasured Disturbances

Use this table to specify the variation of each measured unmeasured disturbance. In the example below, all would be constant at 0.0.

Name	Units	Type	Initial Value	Size	Time	Period
Feed Rate	kmol/min	Constant	0.0			
Distillate Purity	mol %	Constant	0.0			
Bottoms Purity	mol %	Constant	0.0			
Reflux Rate	kmol/min	Constant	0.0			

Unmeasured Disturbance Locations. You can simulate an unmeasured disturbance in any of the following locations:

- The plant's unmeasured disturbance (UD) inputs (if any)
- The plant's measured outputs (MO)
- The plant's manipulated variable (MV) inputs

All of the above will appear as rows in the table. In the case of a measured output or manipulated variable, the disturbance is an additive bias.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-18. Any changes apply to the entire design.)

The **Type** column specifies the disturbance variation. To change this, click the cell and select a choice from the resulting menu.

The significance of the **Initial Value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-60.

Open-Loop Simulations. For open-loop simulations, you can vary the MV unmeasured disturbance to simulate the plant's response to a particular MV. The MV signal coming from the controller stays at its nominal value, and the MV unmeasured disturbance adds to it.

For example, suppose Reflux Rate is an MV, and the corresponding row in the table below represents an unmeasured disturbance in this MV.

Unmeasured disturbances

Name	Units	Type	Initial Value	Size	Time	Period
Feed Rate	kmol/min	Constant	0.0			
Distillate Purity	mol %	Constant	0.0			
Bottoms Purity	mol %	Constant	0.0			
Reflux Rate	kmol/min	Constant	0.0			

You could set it to a constant value of 1 to simulate the plant's open-loop unit-step response to the Reflux Rate input. (In a closed-loop simulation, controller adjustments would also contribute, changing the response.)

Similarly, an unmeasured disturbance in an MO adds to the output signal coming from the plant. If there are no changes at the plant input, the plant outputs are constant, and you see only the change due to the disturbance. This allows you to check the disturbance character before running a closed-loop simulation.

Signal Type Settings

The table below is an example that uses five of the six available signal types (the **Constant** option has been illustrated above). The cells with white backgrounds are the entries you must supply. All have defaults.

Name	Units	Type	Initial Value	Size	Time	Period
Distillate Purity	mol %	Step	0.0	1.0	1.0	
Bottoms Purity	mol %	Ramp	0.0	1.0	1.0	
Reflux Rate	kmol/min	Sine	0.0	1.0	0.0	1.0
Steam Rate	kmol/min	Pulse	0.0	1.0	0.0	1.0
Feed Rate	kmol/min	Gaussian	0.0	1.0	1.0	

Constant. The signal will be held at the specified **Initial Value** for the entire simulation.

$y = y_0$ for $t \geq 0$.

Step. Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal changes step-wise by **Size** units. Its value thereafter = **Initial Value + Size**.

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M$ for $t > 0$, where M = **Size**.

Ramp. Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal begins to vary linearly with slope **Size**.

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M(t - t_0)$ for $t \geq t_0$, where M = **Size**.

Sine. Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal begins to vary sinusoidally with amplitude **Size** and period **Period**.

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M\sin[\omega(t - t_0)]$ for $t \geq t_0$, where $M =$ **Size**, $\omega = 2\pi/$**Period**.

Pulse. Prior to **Time**, the signal = **Initial Value**. At **Time**, a square pulse of duration **Period** and magnitude **Size** occurs.

$y = y_0$ for $0 \leq t < t_0$, where $y_0 =$ **Initial Value**, $t_0 =$ **Time**

$y = y_0 + M$ for $t_0 \leq t \leq t_0$, where $M =$ **Size**, $T =$ **Period**

$y = y_0$ for $t \geq t_0 + T$

Gaussian. Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal begins to vary randomly about **Initial Value** with standard deviation **Size**.

$y = y_0$ for $0 \leq t < t_0$, where $y_0 =$ **Initial Value**, $t_0 =$ **Time**

$y = y_0 + M$randn for $t \geq t_0$, where $M =$ **Size**.

randn is the MATLAB random-normal function, which generates random numbers having zero mean and unit variance.

Simulation Button

Click the **Simulate** button to simulate the scenario. You can also press **Ctrl+R**, use the toolbar icon (see "Toolbar" on page 5-6), or use the **MPC/Simulate** menu option (see "Menu Bar" on page 5-4).

Tuning Advisor Button

Click **Tuning Advisor** to open a window that helps you improve your controller's performance. See "Analyze Sensitivity Using the Tuning Advisor" on page 5-63.

Right-Click Menus

Copy Scenario. Creates a new simulation scenario having the same settings and a default name.

Delete Scenario. Deletes the scenario.

Rename Scenario. Opens a dialog box allowing you to rename the scenario.

Note Each scenario in a design project/task must have a unique name.

Analyze Sensitivity Using the Tuning Advisor

In this section...
"Defining the Performance Metric" on page 5-65
"Baseline Performance" on page 5-67
"Sensitivities and Tuning Advice" on page 5-67
"Updating the Controller" on page 5-70
"Restoring Baseline Tuning" on page 5-70
"Modal Dialog Behavior" on page 5-70
"Scenarios for Performance Measurement" on page 5-70

When you design MPC controllers, you can use the Tuning Advisor to help you determine which weight has the most influence on the closed-loop performance. The Tuning Advisor also helps you determine in which direction to change the weight to improve performance. Using the Advisor, you can know numerically how each weight impacts the closed-loop performance, which makes designing MPC controllers easier when the closed-loop responses does not depend intuitively on the weights.

To start the Tuning Advisor, click **Tuning Advisor** in a simulation scenario view (see "Tuning Advisor Button" on page 5-61). The next figure shows the default Tuning Advisor window for a distillation process in which there are two controlled outputs, two manipulated variables, and one measured disturbance (which the Tuning Advisor ignores). In this case, the originating scenario is **Scenario1**.

The Tuning Advisor populates the **Current Tuning** column with the most recent tuning weights of the controller displayed in the **Controller in Design**. In this case, Obj is the controller. The Advisor also initializes the **Performance Weight** column to the same values. The **Scenario in Design** displays the scenario from which you started the Tuning Advisor. The Advisor uses this scenario to evaluate the controller's performance.

The columns highlighted in grey are Tuning Advisor displays and are read-only. For example, signal names come from the "Signal Definition View" on page 5-18 and are blank unless you defined them there.

To tune the weights using the Tuning Advisor:

1 Specify the performance metric.

2 Compute the baseline performance.

3 Adjust the weights based on the computed sensitivities.

4 Recompute the performance metric.

5 Update the controller

Defining the Performance Metric

In order to obtain tuning advice, you must first provide a quantitative scalar performance measure, *J*.

Select the Performance Function

Select a performance metrics from the **Select a performance function** drop-down list in the upper right-hand corner of the Advisor. You can choose one of four standard ways to compute the performance measure, *J*. In each case, the goal is to minimize *J*.

- ISE (Integral of Squared Error, the default). This is the standard linear quadratic weighting of setpoint tracking errors, manipulated variable movements, and deviations of manipulated variables from targets (if any). The formula is

$$J = \sum_{i=1}^{Tstop} \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

  where `Tstop` is the number of controller sampling intervals in the scenario, $_{yij}$ is the deviation of output *j* from its setpoint (reference) at time step *i*, e_{uij} is the deviation of manipulated variable *j* from its target at time step *i*, Δu_{ij} is the change in manipulated variable *j* at time step *i* (i.e., $\Delta u_{ij} = u_{ij} - u_{i-1,j}$),

  and w_j^y, w_j^u, and $w_j^{\Delta u}$ are non-negative *performance weights*.

- IAE (Integral of Absolute Error). Similar to the ISE but with squared terms replaced by absolute values

$$J = \sum_{i=1}^{Tstop} \left(\sum_{j=1}^{n_y} |w_j^y e_{yij}| + \sum_{j=1}^{n_u} (|w_j^u e_{uij}| + |w_j^{\Delta u} \Delta u_{ij}|) \right)$$

  The IAE gives less emphasis to any large deviations.

- ITSE (time-weighted Integral of Squared Errors)

$$J = \sum_{i=1}^{Tstop} i\Delta t \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

which penalizes deviations at long times more heavily than the ISE, i.e., it favors controllers that rapidly eliminate steady-state offset.

- ITAE (time-weighted Integral of Absolute Errors)

$$J = \sum_{i=1}^{Tstop} i\Delta t \left(\sum_{j=1}^{n_y} |w_j^y e_{yij}| + \sum_{j=1}^{n_u} (|w_j^u e_{uij}| + |w_j^{\Delta u} \Delta u_{ij}|) \right)$$

which is like the ITSE but with less emphasis on large deviations.

Specify Performance Weights

Each of the above formulae use the same three performance weights, w_j^y, w_j^u, and $w_j^{\Delta u}$. All must be non-negative real numbers. Use the weights to:

- Eliminate a term by setting its weight to zero. For example, a manipulated

  variable rarely has a target value, in which case you should set its w_j^u to zero. Similarly if a plant output is monitored but doesn't have a setpoint,

  set its w_j^y to zero.

- Scale the variables so their absolute or squared errors influence J appropriately. For example, an e_{yij} of 0.01 in one output might be as important as a value of 100 in another. If you have chosen the ISE, the first should have a weight of 100 and the second 0.01. In other words, scale all equally important expected errors to be of order unity.

  A Model Predictive Controller uses weights internally as tuning devices. Although there is some common ground, the performance weights and tuning weights should differ in most cases. Choose performance weights to define good performance and then tune the controller weights to achieve it. The Tuning Advisor's main purpose is to make this task easier.

Baseline Performance

After you define the performance metric and specify the performance weights, compute a baseline J for the scenario by clicking **Baseline**. The next figure

shows how this transforms the above example (the two $w_j^{\Delta u}$ performance weights have also been set to zero because manipulated variable changes are acceptable if needed to achieve good setpoint tracking for the two (equally weighted) outputs. The computed $J = 3.435$ is displayed in **Baseline Performance**, to the right of the **Baseline** button.

The Tuning Advisor also displays response plots for the scenario with the baseline controller (not shown but discussed in "Customize Response Plots" on page 5-71).

Sensitivities and Tuning Advice

Click **Analyze** ito compute the sensitivities, as shown in the next figure. The columns labeled **Sensitivity** and **Tuning Direction** now contain advice.
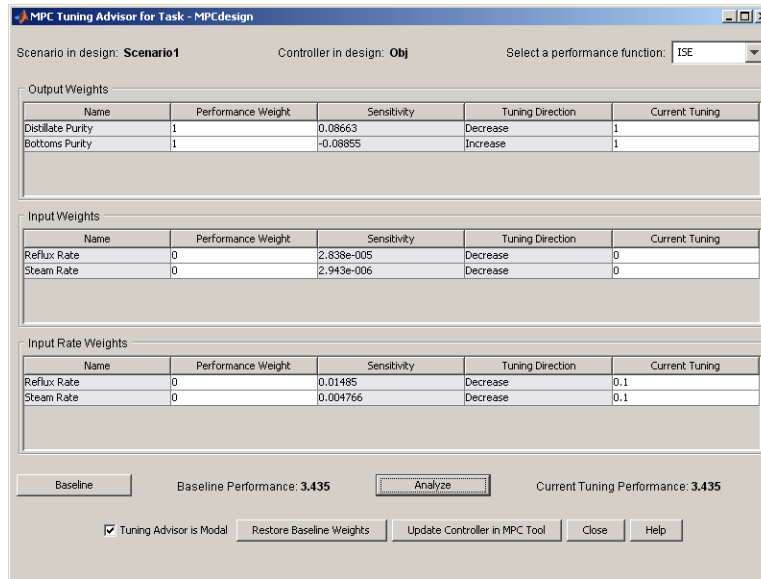
Each sensitivity value is the partial derivative of J with respect to the controller tuning weight in the last entry of the same row. For example, the first output has a sensitivity of 0.08663. If we could assume linearity, a 1-unit increase in this tuning weight, currently equal to 1, would increase J by 0.08663 units. Since we want to minimize J, we should decrease the tuning weight, as suggested by the **Tuning Direction** entry.

The challenge is to choose an adjustment magnitude. The behavior is nonlinear so the sensitivity value is just a rough indication of the likely impact.

You must also consider the tuning weight's current magnitude. For example, if the current value were 0.01, a 1-unit increase would be extreme and a 1-unit decrease impossible, whereas if it were 1000, a 1-unit change would be insignificant.

It's best to focus on a small subset of the tuning weights for which the sensitivities suggest good possibilities for improvement.

In the above example, the $w_j^{\Delta u}$ are poor candidates. The maximum possible change in the suggested direction (decrease) is 0.1, and the sensitivities indicate that this would have a negligible impact on J. The w_j^u are already zero and can't be decreased.

The w_j^y are the only tuning weights worth considering. Again, it seems unlikely that a change will help much. The display below shows the effect of doubling the tuning weight on the bottoms purity (second) output. Note the 2 in the last column of this row. After you click **Analyze,** the response plots (not shown) make it clear that this output tracks its setpoint more accurately but at the expense of the other, and the overall J actually increases.

Notice also that the sensitivities have been recomputed with respect to the revised controller tuning weights. Again, there are no obvious opportunities for improved performance.

Thus, we have quickly determined that the default controller tuning weights are near-optimal in this case, and further tuning is not worth the effort.

Updating the Controller

If you decide a set of modified tuning weights is significantly better than the baseline set, click **Update Controller in MPC Tool**. The tuning weights in the Advisor's last column permanently replace those stored in the **Controller in Design** and become the new baseline. All displays update accordingly.

Restoring Baseline Tuning

If you click **Restore Baseline Weights**, the Advisor will revert to the most recent baseline condition.

Modal Dialog Behavior

By default, the Advisor window is modal, meaning that you won't be able to access any other MATLABwindows while the Advisor is active. You can disable **Tuning Advisor is Modal**, as shown in the above example. This is *not recommended*, however. In particular, if you return to the Design Tool and modify your controller, your changes won't be communicated to the Advisor. Instead, close the Advisor, modify the controller and then re-open the Advisor.

Scenarios for Performance Measurement

The scenario used with the Advisor should be a true test of controller performance. It should include a sequence of typical setpoint changes and disturbances. It is also good practice to test controller robustness with respect to prediction model error. The usual approach is to define a scenario in which the plant being controlled differs from the controller's prediction model.

Customize Response Plots

Each time you simulate a scenario, the design tool plots the corresponding plant input and output responses. The graphic below shows such a *response plot* for a plant having two outputs (the corresponding input response plot is not shown).
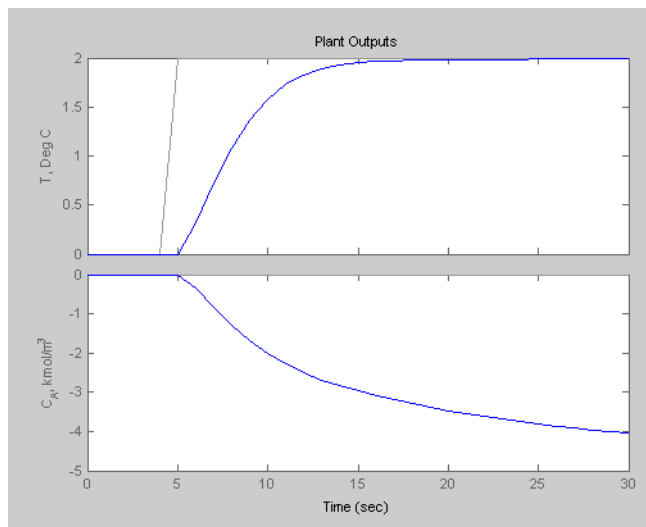
By default, each plant signal plots in its own graph area (as shown above). If the simulation is closed loop, each output signal plot include the corresponding setpoint.

The following sections describe response plot customization options:

- "Data Markers" on page 5-72
- "Displaying Multiple Scenarios" on page 5-73
- "Viewing Selected Variables" on page 5-74
- "Grouping Variables in a Single Plot" on page 5-75
- "Normalizing Response Amplitudes" on page 5-75

Data Markers

You can use data markers to label a curve or to display numerical details.

Adding a Data Marker

To add a data marker, click the desired curve at the location you want to mark. The following graph shows a marker added to each output response and its corresponding setpoint.
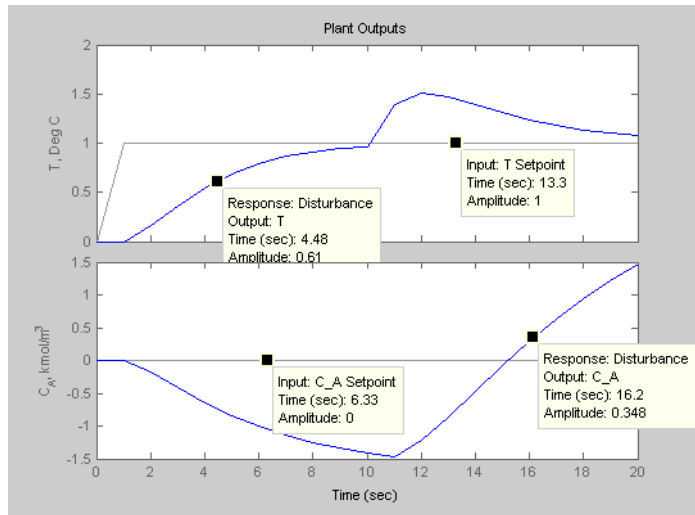
Data Marker Contents

Each data marker provides information about the selected point, as follows:

- **Response** – The *scenario* that generated the curve.
- **Time** – The time value at the data marker location.
- **Amplitude** – The signal value at the data marker location.
- **Output** – The plant variable name (plant outputs only).
- **Input** – Variable name for plant inputs and setpoints.

Changing a Data Marker's Alignment

To relocate the data marker's label (without moving the marker), right-click the marker, and select one of the four **Alignment** menu options. The above example shows three of the possible four alignment options.

Relocating a Data Marker

To move a marker, left-click it (holding down the mouse key) and drag it along its curve to the desired location.

Deleting Data Markers

To delete all data markers in a plot, click in the plot's white space.

To delete a single data marker, right-click it and select the **Delete** option.

Right-Click Options

Right-click a data marker to use one of the following options:

- **Alignment** – Relocate the marker's label.
- **Font Size** – Change the label's font size.
- **Movable** – On/off option that makes the marker movable or fixed.
- **Delete** – Deletes the selected marker.
- **Interpolation** – Interplolate linearly between the curve's data points, or locate at the nearest data point.
- **Track Mode** – Changes the way the marker responds when you drag it.

Displaying Multiple Scenarios

By default the response plots include all the scenarios you've simulated. The example below shows a response plot for a plant with two outputs. The data markers indicate the two scenarios being plotted: "Accurate Model" and "Perturbed Model". Both scenarios use the same setpoints (not marked—the lighter solid lines).

Viewing Selected Scenarios

If your plots are too cluttered, you can hide selected scenarios. To do so:

- Right-click in the plot's white space.

- Select **Responses** from the resulting context menu.

- Toggle a response on or off using the submenu.

Note This selection affects all variables being plotted.
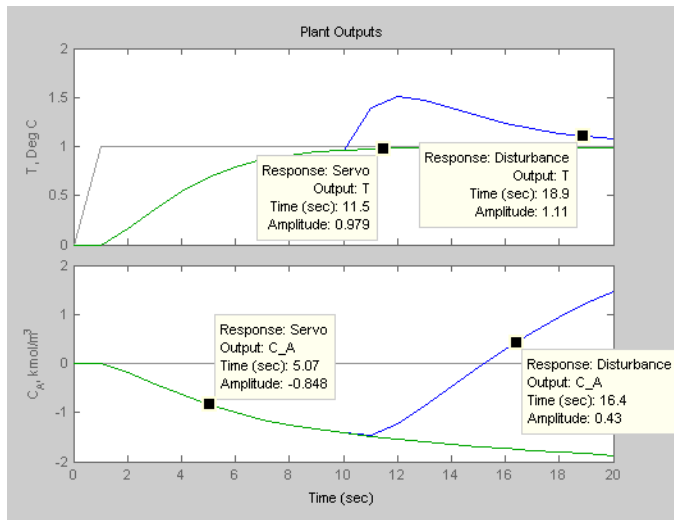
Revising a Scenario

If you modify and recalculate a scenario, its data are replotted, replacing the original curves.

Viewing Selected Variables

By default, the design tool plots all plant inputs in a single window, and plots all plant outputs in another. If your application involves many signals, the plots of each may be too small to view comfortably.

Therefore, you can control the variables being plotted. To do so, right-click in a plot's white space and select **Channel Selector** from the resulting menu. A dialog box appears, on which you can opt to show or hide each variable.

Grouping Variables in a Single Plot

By default, each variable appears in its own plot area. You can instead choose to display variables together in a single plot. To do so, right-click in a plot's white space, and select **Channel Grouping**, and then select **All**.

To return to the default mode, use the **Channel Grouping**: **None** option.

Normalizing Response Amplitudes

When you're using the **Channel Grouping**: **All** option, you might find that the variables have very different scales, making it difficult to view them together. You can choose to *normalize* the curves, so that each expands or contracts to fill the available plot area.

For example, the plot below shows two plant outputs together (**Channel Grouping**: **All** option). The outputs have very different magnitudes. When plotted together, it's hard to see much detail in the smaller response.
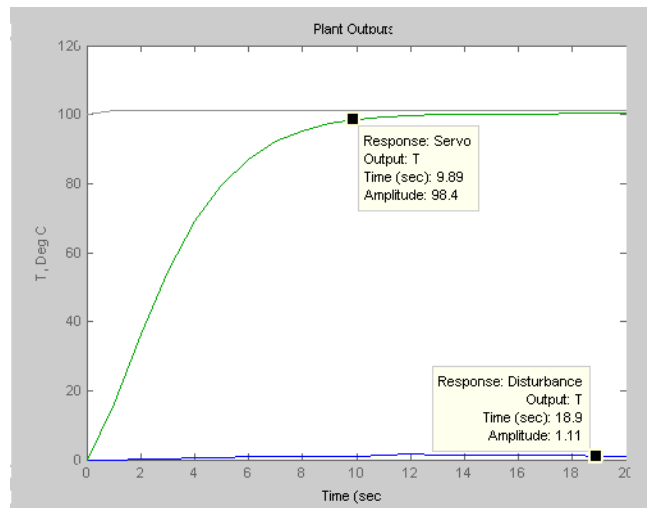
The plot below shows the normalized version, which displays each curve's variations clearly.

The y-axis scale is no longer meaningful, however. If you want to know a normalized signal's amplitude, use a data marker (see "Adding a Data

Marker" on page 5-72). Note that the two data markers on the plot below are at the same normalized y-axis location, but correspond to very different amplitudes in the original (unnormalized) coordinates.

Index