Model Predictive Control Toolbox™ User’s Guide


The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.
**Revision History**

<table>
<thead>
<tr>
<th>Date</th>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2004</td>
<td>First printing</td>
<td>New for Version 2.1 (Release 14SP1)</td>
</tr>
<tr>
<td>March 2005</td>
<td>Online only</td>
<td>Revised for Version 2.2 (Release 14SP2)</td>
</tr>
<tr>
<td>September 2005</td>
<td>Online only</td>
<td>Revised for Version 2.2.1 (Release 14SP3)</td>
</tr>
<tr>
<td>March 2006</td>
<td>Online only</td>
<td>Revised for Version 2.2.2 (Release 2006a)</td>
</tr>
<tr>
<td>September 2006</td>
<td>Online only</td>
<td>Revised for Version 2.2.3 (Release 2006b)</td>
</tr>
<tr>
<td>March 2007</td>
<td>Online only</td>
<td>Revised for Version 2.2.4 (Release 2007a)</td>
</tr>
<tr>
<td>September 2007</td>
<td>Online only</td>
<td>Revised for Version 2.3 (Release 2007b)</td>
</tr>
<tr>
<td>March 2008</td>
<td>Online only</td>
<td>Revised for Version 2.3.1 (Release 2008a)</td>
</tr>
<tr>
<td>October 2008</td>
<td>Online only</td>
<td>Revised for Version 3.0 (Release 2008b)</td>
</tr>
<tr>
<td>March 2009</td>
<td>Online only</td>
<td>Revised for Version 3.1 (Release 2009a)</td>
</tr>
<tr>
<td>September 2009</td>
<td>Online only</td>
<td>Revised for Version 3.1.1 (Release 2009b)</td>
</tr>
<tr>
<td>March 2010</td>
<td>Online only</td>
<td>Revised for Version 3.2 (Release 2010a)</td>
</tr>
<tr>
<td>September 2010</td>
<td>Online only</td>
<td>Revised for Version 3.2.1 (Release 2010b)</td>
</tr>
</tbody>
</table>
# Contents

## Introduction

1

- Model Predictive Control of a SISO Plant .......................... 1-2
  - Typical Sampling Instant .................................. 1-5
  - Prediction and Control Horizons ............................ 1-8

- MIMO Plants .............................................. 1-10
  - Optimization and Constraints ............................... 1-10
  - Estimating States from Measured Data ........................ 1-14
  - Blocking .................................................. 1-14

## Model Predictive Control Problem Setup

2

- Prediction Model ............................................. 2-2
  - Offsets ................................................... 2-4

- Optimization Problem .......................................... 2-5
  - Standard Form ............................................. 2-5
  - Alternative Cost Function .................................. 2-7
  - Remarks on the Constraint Formulation ...................... 2-8

- State Estimation ............................................... 2-9
  - Measurement Noise Model .................................... 2-9
  - Output Disturbance Model ................................... 2-10
  - State Observer ............................................. 2-10

- QP Matrices .................................................... 2-13
  - Prediction .................................................... 2-13
  - Optimization Variables ..................................... 2-14
  - Cost Function ............................................... 2-16
  - Constraints .................................................. 2-17
Reference for the Design Tool GUI

5

Opening the MPC Design Tool .......................... 5-2

Menu Bar ........................................... 5-4
  File Menu ....................................... 5-4
  MPC Menu ....................................... 5-5

Toolbar ........................................... 5-7

Tree View ........................................... 5-8
  Node Types ..................................... 5-8
  Renaming a Node ................................. 5-8

Importing a Plant Model .............................. 5-10
  Import from .................................... 5-11
  Import to ....................................... 5-12
  Buttons ......................................... 5-12
  Importing a Linearized Plant Model ............... 5-13

Importing a Controller ............................... 5-16
  Import from .................................... 5-16
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import to</td>
<td>5-18</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-18</td>
</tr>
<tr>
<td><strong>Exporting a Controller</strong></td>
<td>5-19</td>
</tr>
<tr>
<td>Dialog Box Options</td>
<td>5-19</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-20</td>
</tr>
<tr>
<td><strong>Signal Definition View</strong></td>
<td>5-21</td>
</tr>
<tr>
<td>MPC Structure Overview</td>
<td>5-22</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-22</td>
</tr>
<tr>
<td>Signal Properties Tables</td>
<td>5-22</td>
</tr>
<tr>
<td>Right-Click Menu Options</td>
<td>5-24</td>
</tr>
<tr>
<td><strong>Plant Models View</strong></td>
<td>5-26</td>
</tr>
<tr>
<td>Plant Models List</td>
<td>5-27</td>
</tr>
<tr>
<td>Model Details</td>
<td>5-27</td>
</tr>
<tr>
<td>Additional Notes</td>
<td>5-28</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-28</td>
</tr>
<tr>
<td>Right-Click Options</td>
<td>5-28</td>
</tr>
<tr>
<td><strong>Controllers View</strong></td>
<td>5-29</td>
</tr>
<tr>
<td>Controllers List</td>
<td>5-30</td>
</tr>
<tr>
<td>Controller Details</td>
<td>5-30</td>
</tr>
<tr>
<td>Additional Notes</td>
<td>5-31</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-31</td>
</tr>
<tr>
<td>Right-Click Options</td>
<td>5-32</td>
</tr>
<tr>
<td><strong>Simulation Scenarios List</strong></td>
<td>5-33</td>
</tr>
<tr>
<td>Scenarios List</td>
<td>5-34</td>
</tr>
<tr>
<td>Scenario Details</td>
<td>5-35</td>
</tr>
<tr>
<td>Additional Notes</td>
<td>5-35</td>
</tr>
<tr>
<td>Buttons</td>
<td>5-35</td>
</tr>
<tr>
<td>Right-Click Options</td>
<td>5-35</td>
</tr>
<tr>
<td><strong>Controller Specifications View</strong></td>
<td>5-36</td>
</tr>
<tr>
<td>Model and Horizons Tab</td>
<td>5-37</td>
</tr>
<tr>
<td>Constraints Tab</td>
<td>5-40</td>
</tr>
<tr>
<td>Constraint Softening</td>
<td>5-43</td>
</tr>
<tr>
<td>Weight Tuning Tab</td>
<td>5-46</td>
</tr>
<tr>
<td>Entering Vectors in Table Cells</td>
<td>5-49</td>
</tr>
<tr>
<td>Estimation Tab</td>
<td>5-49</td>
</tr>
</tbody>
</table>
Contents
Introduction

- “Model Predictive Control of a SISO Plant” on page 1-2
- “MIMO Plants” on page 1-10
Model Predictive Control of a SISO Plant

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

Consider instead the simpler application shown in the following figure, Block Diagram of a SISO Model Predictive Control Toolbox™ Application on page 1-2. See the nomenclature summary in the table, Model Predictive Control Toolbox™ Signals on page 1-3. This plant could 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, \( y \), 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 SISO plant actually has multiple inputs, as shown in the following figure. In addition to the manipulated variable input, \( u \), there may be a measured disturbance, \( v \), and an unmeasured disturbance, \( d \).

Block Diagram of a SISO Model Predictive Control Toolbox™ Application

The unmeasured disturbance is always present. As shown in the previous figure, 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, $y$. This output is fed back to the controller as shown in the Block Diagram of a SISO Model Predictive Control Toolbox™ Application on page 1-2 figure.

**Model Predictive Control Toolbox Signals**

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

Some applications have unmeasured disturbances only. A *measured* disturbance, $v$, is another independent input affecting $\bar{y}$. In contrast to $d$, the controller receives the measured $v$ directly, as shown in the figure Block Diagram of a SISO Model Predictive Control Toolbox™ Application on page 1-2. This 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 is called *feedforward* control.
In other words, Model Predictive Control Toolbox design always provides *feedback* compensation for unmeasured disturbances and *feedforward* compensation for any measured disturbance.

Model Predictive Control Toolbox 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). One may also specify bounds on $\bar{y}$. These constraint specifications are a distinguishing feature of Model Predictive Control Toolbox design and can be particularly valuable when one has 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 an *anti-windup* feature.

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. They could vary randomly with a zero mean, or could exhibit a non-zero, 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 above feedforward/feedback actions comprise the controller’s *regulator* mode. Model Predictive Control Toolbox 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 $u \rightarrow \tilde{y}$ model, and whether or not future setpoint variations can be anticipated, i.e., known in advance. If so, it provides feedforward compensation for these.

**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, $\Delta t$ (also called the control interval). This section provides more details on the events occurring at each sampling instant.
Controller State at the \( k \)th Sampling Instant

The figure, Controller State at the \( k \)th Sampling Instant on page 1-6, 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, \( y_{k-1}, y_{k-2}, \ldots \), are known and are the filled circles in the figure Controller.
State at the \( k \)th Sampling Instant on page 1-6 (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 1-6 (b) shows the controller’s previous moves, \( u_{k-4}, \ldots, u_{k-1} \), 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 1-6 (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, \( \bar{y}_{k+1}, \ldots, \bar{y}_{k+P} \). To accomplish this, the controller uses all past and current measurements and the models \( u \to \bar{y}, d \to \bar{y}, w \to \bar{y}, \) and \( z \to y \). For details, see “Prediction” on page 2-13 and “State Estimation” on page 2-9.

2. **Optimization.** Values of setpoints, measured disturbances, and constraints are specified over a finite horizon of future sampling instants, \( k+1, k+2, \ldots, k+P \), where \( P \) (a finite integer \( \geq 1 \)) is the prediction horizon —see figure Controller State at the \( k \)th Sampling Instant on page 1-6 (a). The controller computes \( M \) moves \( u_k, u_{k+1}, \ldots, u_{k+M-1} \), where \( M \) (\( \geq 1, \leq P \)) is the control horizon—see figure Controller State at the \( k \)th Sampling Instant on page 1-6 (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 1-6 (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 1-6 (a). Notice that both are within their constraints, \( u_{\min} \leq u_{k+i} \leq u_{\max} \) 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, \( \Delta t \) 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.

![Diagram](image)

- **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.
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 $y$.

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 \( w_j^y \gg w_{i\neq j}^y \), 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.

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 float 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:

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

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 \( w_{\Delta u}^{j} \) is a weight, which must be zero or positive. Increasing \( w_{\Delta u}^{j} \) forces the controller to make smaller, more cautious \( \Delta u_j \) 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 steppoint 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 \( j^{th} \) 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 2-9.

**Blocking**

In figure Controller State at the $k$th Sampling Instant on page 1-6 (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 1-15 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 $P >> M$ (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.
Model Predictive Control
Problem Setup

- “Prediction Model” on page 2-2
- “Optimization Problem” on page 2-5
- “State Estimation” on page 2-9
- “QP Matrices” on page 2-13
- “Model Predictive Control Computation” on page 2-19
- “Using Identified Models” on page 2-20
**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 2-10 and `setindist`).

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

\[
x(k+1) = Ax(k) + B_u u(k) B_v v(k) + B_d d(k)
\]

\[
y_m(k) = C_m x(k) + D_{vm} v(k) + D_{dm} d(k)
\]

\[
y_u(k) = C_u x(k) + D_{vu} v(k) + D_{du} d(k)
\]
where \( x(k) \) is the \( n_x \)-dimensional state vector of the plant, \( u(k) \) is the \( n_u \)-dimensional vector of manipulated variables (MV), i.e., the command inputs, \( v(k) \) is the \( n_v \)-dimensional vector of measured disturbances (MD), \( d(k) \) is the \( n_d \)-dimensional vector of unmeasured disturbances (UD) entering the plant, \( y_m(k) \) is the vector of measured outputs (MO), and \( y_u(k) \) is the vector of unmeasured outputs (UO). The overall \( n_y \)-dimensional output vector \( y(k) \) collects \( y_m(k) \) and \( y_u(k) \).

Model Predictive Control Toolbox software accepts both plant models specified as LTI objects, and models obtained from input/output data using System Identification Toolbox™ (IDMODEL objects), see “Using Identified Models” on page 2-20.

In the above equations \( d(k) \) collects both 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 matrices \( A, B, C, D \) of the model representing the process to control 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 \texttt{trim} on a Simulink® model describing the nonlinear dynamical equations, and \( A, B, C, D \) by using \texttt{linmod}. The linearized model has the form:

\[
x' \approx f(x_0, u_0, v_0, d_0) + \nabla_x f(x_0, u_0, v_0, d_0)(x - x_0) + \nabla_x f(x_0, u_0, v_0, d_0)(u - u_0)
\]

\[
+ \nabla_v f(x_0, u_0, v_0, d_0)(v - v_0) + \nabla_d f(x_0, u_0, v_0, d_0)(d - d_0)
\]

\[
y \approx h(x_0, u_0, v_0, d_0) + \nabla_x h(x_0, u_0, v_0, d_0)(x - x_0) + \nabla_u h(x_0, u_0, v_0, d_0)(u - u_0)
\]

\[
+ \nabla_v h(x_0, u_0, v_0, d_0)(v - v_0) + \nabla_d h(x_0, u_0, v_0, d_0)(d - d_0)
\]

The matrices \( A, B, C, D \) of the model 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

Standard Form
Assume that the estimates of $x(k)$, $x_d(k)$ are available at time $k$ (for state estimation, see “State Estimation” on page 2-9). The model predictive control action at time $k$ is obtained by solving the optimization problem

$$\min_{\Delta u(k \mid k), \ldots, \Delta u(m-1+k \mid k), \varepsilon} \left[ \sum_{i=0}^{p-1} \left( \sum_{j=1}^{n} |e_{i+1,j}^y (y_j (k+i+1 \mid k) - r_j (k+i+1))|^2 + \sum_{j=1}^{n} |w_{i,j}^u \Delta u_j (k+i \mid k)|^2 + \sum_{j=1}^{n} |e_{i,j}^u (u_j (k+i \mid k) - u_{j\text{target}} (k+i))|^2 \right) + p \varepsilon^2 \right]$$

where the subscript “( )” denotes the $j$-th component of a vector, “(k+i | k)” denotes the value predicted for time $k+i$ based on the information available at time $k$; $r(k)$ is the current sample of the output reference, subject to

$$u_{j\text{min}} (i) - \varepsilon V_{j\text{min}}^u (i) \leq u_j (k+i \mid k) \leq u_{j\text{max}} (i) + \varepsilon V_{j\text{max}}^u (i)$$

$$\Delta u_{j\text{min}} (i) - \varepsilon V_{j\text{min}}^\Delta u (i) \leq \Delta u_j (k+i \mid k) \leq \Delta u_{j\text{max}} (i) + \varepsilon V_{j\text{max}}^\Delta u (i)$$

$$y_{j\text{min}} (i) - \varepsilon V_{j\text{min}}^y (i) \leq y_j (k+i+1 \mid k) \leq y_{j\text{max}} (i) + \varepsilon V_{j\text{max}}^y (i)$$

$$\Delta u (k+h \mid k) = 0$$

$$\varepsilon \geq 0$$

where

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

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

with respect to the sequence of input increments $\{\Delta u (k \mid k), \ldots, \Delta u (m-1+k \mid k)\}$ and to the slack variable $\varepsilon$, and by setting $u(k)=u(k-1)+\Delta u(k \mid k)^*$, where $\Delta u(k \mid k)^*$ is the first element of the optimal sequence.

Note Although only the measured output vector $y_m (k)$ is fed back to the model predictive controller, $r(k)$ 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*). A similar anticipative action can be performed with respect to measured disturbances $v(k)$, namely $v(k+i)=v(k)$ if the measured disturbance is not known in advance (e.g., is coming from a Simulink block) or $v(k+i)$ is obtained from the workspace. In the prediction, $d(k+i)$ is instead obtained by setting $n_d(k+i)=0$ in figures Block Diagram of a SISO Model Predictive Control Toolbox™ Application on page 1-2 and Controller State at the $k$th Sampling Instant on page 1-6.

$w^u_{i,j}, w^u_{i,j}, w^y_{i,j}$ are nonnegative weights for the corresponding variable. The smaller $w$, the less important is the behavior of the corresponding variable to the overall performance index.

$u_{j,\text{min}}, u_{j,\text{max}}, \Delta u_{j,\text{min}}, \Delta u_{j,\text{max}}, y_{j,\text{min}}, y_{j,\text{max}}$ are lower/upper bounds on the corresponding variables. In Equation 2-4, the constraints on $u$, $\Delta u$, and $y$ are relaxed by introducing the slack variable $\varepsilon \geq 0$. The weight $\rho\varepsilon$ on the slack variable $\varepsilon$ penalizes the violation of the constraints. The larger $\rho\varepsilon$ with respect to input and output weights, the more the constraint violation is penalized.

The Equal Concern for the Relaxation (ECR) vectors $V^u_{\text{min}}, V^u_{\text{max}}, V^{\Delta u}_{\text{min}}$, $V^D_{\text{max}}, V^y_{\text{min}}, V^y_{\text{max}}$ have nonnegative entries which represent the concern for relaxing the corresponding constraint; the larger $V$, the softer the constraint. $V=0$ means that the constraint is a *hard* one that cannot be violated. By default, all input constraints are hard ($V^u_{\text{min}} = V^u_{\text{max}} = V^{\Delta u}_{\text{min}} = V^{\Delta u}_{\text{max}} = 0$) and all output constraints are soft ($V^y_{\text{min}} = V^y_{\text{max}} = 1$). As hard output constraints may cause infeasibility of the optimization problem (for instance, because of unpredicted disturbances, model mismatch, or just because of numerical round off), a warning message is produced if $V^y_{\text{min}}, V^y_{\text{max}}$ are smaller than a given small value and automatically adjusted at that value. By default,

$$\rho\varepsilon = 10^5 \max \{w^u_{i,j}, w^u_{i,j}, w^y_{i,j}\}.$$  \hspace{1cm} (2-4)

Note that also ECRs can be time varying.
Vector $u_{\text{target}}(k+i)$ is a setpoint for the input vector. One typically uses $u_{\text{target}}$ if the number of inputs is greater than the number of outputs, as a sort of lower-priority setpoint.

As mentioned earlier, only $\Delta u(k \mid k)$ is actually used to compute $u(k)$. The remaining samples $\Delta u(k+i \mid k)$ 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 all the bounds are infinite, then the slack variable $\varepsilon$ is removed, and the problem in Equation 2-3 and Equation 2-4 is solved analytically. Otherwise a Quadratic Programming (QP) solver is used. The matrices associated with the quadratic optimization problem are described in “QP Matrices” on page 2-13.

Since output constraints are always soft, the QP problem is never infeasible. If for numerical reasons the QP problem becomes infeasible, the second sample from the previous optimal sequence is applied, i.e. $u(k)=u(k-1)+\Delta^* u(k \mid k-1)$.

**Note** To improve numerical robustness for constrained model predictive control problems the default value $\Delta u_{\mu, \text{min}}$ for unbounded input rates is $-10$ and the maximum allowed lower bound is $-1e5$. The default value for unconstrained problems is minus infinity.

**Alternative Cost Function**

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

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

$$+ \left[ u(k+i | k) - u_{\text{target}}(k+i) \right] \left[ u(k+i | k) - u_{\text{target}}(k+i) \right]^T R_u \left[ u(k+i | k) - u_{\text{target}}(k+i) \right] + \rho \varepsilon^2$$

where $Q$ is an $n_y$ by $n_y$ matrix, and $R_{\Delta u}$ and $R_u$ are $n_u$ by $n_u$ matrices, all positive semi-definite. Equation 2-5 allows non-zero 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 the weights \( w_{i,j}^y \), \( w_{i,j}^{\Delta u} \), and \( w_{i,j}^u \) are constant for all \( i = 1, \ldots, p \), and when the 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.

**Remarks on the Constraint Formulation**

The constraints listed in Equation 2-3 are bounds of various types. They are general in the sense that each bound can vary at each step in the prediction horizon. In addition, the linear model relating the plant inputs and outputs is a set of linear equality constraints and is included implicitly.

Quadratic programming allows general equalities and inequalities that are linear combinations of the decision variables. The current Model Predictive Control Toolbox does not allow you to include such constraints explicitly in your controller design.

If you have a small number of such constraints, consider incorporating them as new variables in your model. For example, suppose you would like a linear combination of selected inputs and outputs to be kept less than or equal to a given constant value. You can define a new model output to represent the linear combination. If the combination involves manipulated variables, include a unit delay (to eliminate direct transfer from a manipulated variable to an output) or declare the combination to be an unmeasured output. Then define a bound on this output in your controller design.
State Estimation

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

Measurement Noise Model

We assume that the measured output vector $y_m(k)$ is corrupted by a measurement noise $m(k)$. The measurement noise $m(t)$ 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.
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, in order to reject constant disturbances due for instance to gain nonlinearities, the output disturbance model is a collection of integrators driven by white noise on measured outputs. 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 the user forces it (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.

### 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, and \( x_m(k) \) is the state of the measurement noise model. The estimates are computed from the measured output \( y_m(k) \) by the linear state observer.
State Estimation

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

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

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

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 \texttt{kalman} in the Control System Toolbox documentation) on the extended model

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

\[
y_m(k) = 
\begin{bmatrix}
C_m & D_{dm} C & \tilde{C}
\end{bmatrix}
\begin{bmatrix}
x(k) \\
x_d(k) \\
x_m(k)
\end{bmatrix} + D_{vm} v(k) + 
\begin{bmatrix}
\tilde{D}_m & \tilde{D}
\end{bmatrix}
\begin{bmatrix}
n_d(k) \\
n_m(k) \\
n_u(k) \\
n_v(k)
\end{bmatrix}
\]  

(2-6)

where \(n_d(k)\) 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 combination of 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 together, 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 that you can use to access and modify properties of the state estimator.
This section describes the matrices associated with the model predictive control optimization problem described in “Optimization Problem” on page 2-5.

- “Prediction” on page 2-13
- “Optimization Variables” on page 2-14
- “Cost Function” on page 2-16
- “Constraints” on page 2-17

**Prediction**
Assume for simplicity that the disturbance model in Equation 2-1 and Equation 2-2 is a unit gain (i.e., \( d(k)=n_d(k) \) is a white Gaussian noise). For simplicity, denote by

\[
\begin{bmatrix}
    x \\
    x_d
\end{bmatrix}, \begin{bmatrix}
    A & B_d \tilde{C} \\
    0 & \tilde{A}
\end{bmatrix}, \begin{bmatrix}
    B_u \\
    B_v \\
    0
\end{bmatrix}, \begin{bmatrix}
    B_d \\
    \tilde{B}
\end{bmatrix}, C \left[ \begin{bmatrix}
    C \\
    D_d \tilde{C}
\end{bmatrix}
\right]
\]

Then, the prediction model given by

\[
x(k+1) = Ax(k) + B_u u(k) + B_v v(k) + B_d n_d(k)
\]

\[
y(k) = Cx(k) + D_v v(k) + D_d n_d(k)
\]

Consider for simplicity the prediction of the future trajectories of the model performed at time \( k=0 \). We set \( n_d(i)=0 \) for all prediction instants \( i \), and obtain

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

which gives
\[ \begin{bmatrix} y(1) \\ \vdots \\ y(p) \end{bmatrix} = S_x x(0) + S_u u(-1) + S_u \begin{bmatrix} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix} + H_v \begin{bmatrix} v(0) \\ \vdots \\ v(p) \end{bmatrix} \]

where

\[ S_x = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^p \end{bmatrix} \in \mathbb{R}^{p n_y \times n_x}, \quad S_u = \begin{bmatrix} CB_u \\ CB_u + CAB_u \\ \vdots \\ \sum_{h=0}^{p-1} CA^h B_u \end{bmatrix} \in \mathbb{R}^{p n_y \times n_u} \]

\[ H_v = \begin{bmatrix} CB_v \\ CAB_v \\ \vdots \\ CA^{p-1} B_v \end{bmatrix} \begin{bmatrix} D_v \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{p n_y \times (p+1) n_v}. \]

**Optimization Variables**

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

\[
\begin{bmatrix} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix} = J_M \begin{bmatrix} z_0 \\ \vdots \\ z_{m-1} \end{bmatrix}
\]

where \( J_M \) depends on the choice of blocking moves. Together with the slack variable \( \varepsilon \), vectors \( z_0, \ldots, z_{m-1} \) constitute the free optimization variables of the optimization problem (in case of systems with a single manipulated variables, \( z_0, \ldots, z_{m-1} \) are scalars).
Consider for instance the blocking moves depicted in Blocking Moves: Inputs and Input Increments for moves=[2 3 2] on page 2-15, which corresponds to the choice moves=[2 3 2], or, equivalently, \( u(0)=u(1), \ u(2)=u(3)=u(4), \ u(5)=u(6), \ \Delta u(0)=z_0, \ \Delta u(2)=z_1, \ \Delta u(5)=z_2, \ \Delta u(1)=\Delta u(3)=\Delta u(4)=\Delta u(6)=0. \)

Then, the corresponding matrix \( J_M \) is

\[
J_M = \begin{bmatrix}
I & 0 & 0 \\
0 & 0 & 0 \\
0 & I & 0 \\
0 & 0 & 0 \\
0 & 0 & I \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]
Cost Function

Standard Form
The function to be optimized is

\[
J(z, \varepsilon) = \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right)^T W_u \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right) + \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right)^T W_{\Delta u} \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right) + \rho \varepsilon^2
\]

where

\[
W_u = \text{diag}(w_{0,1}, w_{0,2}, \ldots, w_{0,n_u}, \ldots, w_{p-1,1}, w_{p-1,2}, \ldots, w_{p-1,n_u})
\]

\[
W_{\Delta u} = \text{diag}(w_{\Delta u,0,1}, w_{\Delta u,0,2}, \ldots, w_{\Delta u,0,n_u}, \ldots, w_{\Delta u,p-1,1}, w_{\Delta u,p-1,2}, \ldots, w_{\Delta u,p-1,n_u})
\]

\[
W_y = \text{diag}(w_{y,1,1}, w_{y,1,2}, \ldots, w_{y,1,n_y}, \ldots, w_{y,p,1}, w_{y,p,2}, \ldots, w_{y,p,n_y})
\]

(2-8)

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

\[
J(z, \varepsilon) = \rho \varepsilon^2 + z^T K_{\Delta u} z + 2 \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right)^T K_{\nu} + \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right)^T K_{u} K_{u} + 2 \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right)^T K_{u} \left( \begin{array}{c}
u(0) \\
\vdots \\
u(p-1)
\end{array} \right) + \rho \varepsilon^2
\]

+ constant

(2-9)

**Note** In order to keep the QP problem always strictly convex, if the condition number of the Hessian matrix \( K_{\Delta u} \) is larger than 10^{12}, the quantity \( 10^{*} \text{sqrt}(\varepsilon) \) is added on each diagonal term. This may only occur when all input rates are not weighted (\( W_{\Delta u} = 0 \)) (see “Weights” in the Model Predictive Control Toolbox Reference).
Alternative Cost Function

If the alternative cost function shown in Equation 2-5 is being used, Equation 2-8 is replaced by the following:

\[
\begin{align*}
W_u & = \text{blkdiag}(R_u, \ldots, R_u) \\
W_{\Delta u} & = \text{blkdiag}(R_{\Delta u}, \ldots, R_{\Delta u}) \\
W_y & = \text{blkdiag}(Q, \ldots, Q)
\end{align*}
\] (2-10)

where the block-diagonal matrices repeat \( p \) times, i.e., 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 for more details.

Constraints

Now consider the limits on inputs, input increments, and outputs along with the constraint \( \epsilon \geq 0 \).

\[
\begin{bmatrix}
y_{\text{min}}(1) - \epsilon V_y^y (1) \\
y_{\text{min}}(p) - \epsilon V_y^y (p) \\
u_{\text{min}}(0) - \epsilon V_u^u (0) \\
\vdots \\
u_{\text{min}}(p-1) - \epsilon V_u^u (p-1) \\
\Delta u_{\text{min}}(0) - \epsilon V_{\Delta u}^\Delta u (0) \\
\vdots \\
\Delta u_{\text{min}}(p-1) - \epsilon V_{\Delta u}^\Delta u (p-1)
\end{bmatrix} 
\leq \begin{bmatrix}
y(1) \\
y(p) \\
u(0) \\
\vdots \\
u(p-1) \\
\Delta u(0) \\
\vdots \\
\Delta u(p-1)
\end{bmatrix} 
\leq \begin{bmatrix}
y_{\text{max}}(1) + \epsilon V_y^y (1) \\
y_{\text{max}}(p) + \epsilon V_y^y (p) \\
u_{\text{max}}(0) + \epsilon V_u^u (0) \\
\vdots \\
u_{\text{max}}(p-1) + \epsilon V_u^u (p-1) \\
\Delta u_{\text{max}}(0) + \epsilon V_{\Delta u}^\Delta u (0) \\
\vdots \\
\Delta u_{\text{max}}(p-1) + \epsilon V_{\Delta u}^\Delta u (p-1)
\end{bmatrix}
\]

Note: Upper and lower bounds that are not finite are removed, as well as the input and input-increment bounds over blocked moves.
Similarly to what was done for the cost function, we can substitute $u(k)$, $\Delta u(k)$, $y(k)$, and obtain

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

(2-11)

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

Function `mpc_buildmat` constructs the QP problem matrices.
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`) by using the matrices built at initialization described in “QP Matrices” on page 2-13.

**Unconstrained MPC**

The optimal solution is computed analytically

\[
\begin{align*}
    z^* &= -K_{\Delta u}^{-1} \begin{bmatrix} r(1)^T & \cdots & v(0) \\ \vdots & \ddots & \vdots \\ r(p) & \cdots & v(p) \end{bmatrix} K_r + \begin{bmatrix} u_{\text{target}}(0) \\ \vdots \\ u_{\text{target}}(p-1) \end{bmatrix} K_u + \begin{bmatrix} u_{\text{target}}(0) \\ \vdots \\ u_{\text{target}}(p-1) \end{bmatrix} K_{ut} + x(0)^T K_x
\end{align*}
\]

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

**Constrained Model Predictive Control**

The optimal solution \( z^*, \varepsilon^* \) is computed by solving the quadratic program described in Equation 2-9 and Equation 2-11, using the QP solver coded in the `qpsolver.mex` function (see `qpdantz` for more details).
Using Identified Models

Model Predictive Control Toolbox software is able to handle plant models generated by System Identification Toolbox software from input/output measurements.

Model Predictive Control Toolbox software labels control input signals as ‘Manipulated’, measured input disturbances as ‘Measured’, and unmeasured input disturbances as ‘Unmeasured’. On the other hand, System Identification Toolbox software has a different naming rule, as it calls ‘Measured’ the inputs that are measurable quantities, and ‘Noise’ those that are not.

When you specify an identified model in the Model Predictive Control constructor as the plant model, Model Predictive Control Toolbox software treats ‘Noise’ signals as ‘Unmeasured’ input signals, and ‘Measured’ signals as ‘Manipulated’ signals, assuming that all measured inputs are also manipulated variables. You can later change later signal types, for instance to specify that some measured inputs are measured disturbances, rather than manipulated variables (see `setname`).

Model Predictive Control Toolbox software internally converts the identified model you have provided as a plant model into the classical (A,B,C,D) state-space format. The columns of the B matrix originally related to ‘Noise’ channels are treated as the effect of unmeasured input disturbances on the state of the plant. On the other hand, the columns of the D matrix related to ‘Noise’ channels as treated as the effect of measurement noise superimposed on the output signal. Accordingly, Model Predictive Control Toolbox software treats as the plant model the state-space model obtained from (A,B,C,D) by zeroing the columns of D related to ‘Noise’ channels. Those columns are instead used as a static noise model, or cascaded to an existing noise model if you have specified one. A unit static gain is assumed as the disturbance model, unless you have specified another one.
Model Predictive Control
Simulink Library

- “MPC Library” on page 3-2
- “MPC Controller Block” on page 3-3
- “Multiple MPC Controllers” on page 3-8
MPC Library

The MPC Simulink Library provides two blocks you can use to model MPC control in Simulink.

Access the library using the Simulink Library Browser or by typing `mpclib` at the command prompt. The latter reveals the library’s contents as shown in the following figure.

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 provides a user friendly way to model MPC control in Simulink. It represents a single MPC controller in which the design parameters remain constant throughout a simulation.

**MPC Controller Block Mask**

Once you have placed an MPC Controller block in your Simulink model, you need to specify its properties. Double-click on the block to open its mask. The following figure shows the mask’s default settings.

By default, the **MPC controller** field is empty. Before running a simulation you must enter the name of a valid MPC object. There are two ways to do this:

1. In the **MPC controller** field, type the name of an MPC object that already exists in your workspace. If you want to review the settings before running
a simulation, click the **Design** button to load the named object into the MPC design tool.

2 If your installation includes the Simulink® Control Design™ software, connect the MPC Controller block to the plant it will control (see the following sections for connection details). Leaving the **MPC controller** field empty, click the **Design** button. The MPC Controller block constructs a default MPC object by linearizing the plant defined in the Simulink diagram. See “Importing a Plant Model” on page 5-10 for more information about creating linearized plant models using Model Predictive Control Toolbox software. Refer to the Simulink Control Design documentation for more information about the linearization process.

---

**Note** You can run closed-loop simulations while a controller is being edited in the design tool. In this case, the latest settings from the design tool are used in each the Simulink simulation. This makes it convenient to tune the controller parameters. When you close the design tool, export the final controller design to the workspace so it can be used in subsequent simulations.

---

**Input Signals**

You must connect appropriate Simulink signals to the MPC Controller block’s inputs. 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 3-3, **Enable input port for 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 input port for externally supplied manipulated variables to plant** 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 “Bumpless Transfer in MPC” on page 4-41.

Enable input port for 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 demo `mpcvarbounds` shows how this option works. It enables inports for lower and upper bounds on the manipulated variables (inports \( u_{\min} \) and \( u_{\max} \)) and lower and upper bounds on the controlled outputs (inports \( y_{\min} \) and \( y_{\max} \)). An unconnected constraint inport causes the corresponding variable to be unconstrained.

Enable input port for switching optimization off 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 input port for externally supplied manipulated variables to plant option must also be activated. See “Bumpless Transfer in MPC” on page 4-41 for an example application.

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:

```matlab
  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 demo mpcpreview 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 3-3, 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.

**Using Model Predictive Control Toolbox Software with Real-Time Workshop Software**

After designing an MPC controller in Simulink software using the MPC Controller block, you can use Real-Time Workshop® software to build this controller and deploy it to the following targets for real-time control:

- Generic Real-Time Target
- Real-Time Workshop® Embedded Coder™
• Real-Time Windows Target™
• Rapid Simulation Target
• Target for Freescale™ MPC5xx
• xPC Target™
• dSPACE® Target
• Target for Infineon® TriCore®

The following targets are either not supported or not recommended because they result in significant performance issues:

• Target Support Package™ for Use with TI’s C2000™
• Target Support Package for Use with TI’s C6000™
• Target Support Package for Use with Infineon® C166®
• Tornado® (VxWorks) Real-Time Target

**Note** The Multiple MPC Controllers block has not been tested with the targets supported by Real-Time Workshop software.

The C sources of the S-function executing the MPC Controller block code are available in the `mpcutils/src` folder. You can build a real-time executable by pressing **Ctrl+B** on your Simulink diagram to invoke Real-Time Windows Target and build the model.

In some cases, it is necessary to copy the source files (`mpc_sfun.c`, `mpc_sfun.h`, `mpc_common.c`, `mat_macros.h`, `dantzgmp.h`, `dantzgmp_solver.c`) to a visible folder, such as the current folder `.'`, or `C:\MATLAB\rtw\c\src`. 
Multiple MPC Controllers

The Multiple MPC Controllers (MMPC) block allows you to control a nonlinear plant over a large range of operating points. You can design multiple MPC controllers inside this block, and switch between them in real-time using an input switching signal to the block.

A controller that initially works well can degrade dramatically if the plant is nonlinear and its operating point changes. In conventional feedback control, you might compensate for nonlinear behavior with gain scheduling. Similarly, the MMPC block allows you to transition between multiple MPC controllers in a pre-ordained manner. Each MPC controller is designed to work well in a particular region. When the plant moves away from this operating point, the control system switches to another MPC controller.

As with the standard MPC Controller block described in “MPC Controller Block” on page 3-3, you copy the MMPC block into your Simulink model and then double-click the mask to reveal its settings. Multiple MPC Controller Block Mask Default Settings on page 3-10 shows the default settings.

The key difference between this block and the standard MPC Controller block is the way you designate the controllers to be used. You enter them as an ordered list containing \( N_c \) names, where \( N_c \) is the number of controllers to use and each name designates a valid MPC object existing in the MATLAB workspace. Each named controller must be designed to use the identical set of plant signals. For example, same measured outputs and same manipulated variables.

Use the Add and Delete buttons as needed to add and remove MPC controllers from the list.

**Note** You can also load one or more controllers into the design tool where you can view controller properties and modify them. Select controller(s) using the Design It checkboxes. Then click the Design button, which causes the design tool to open and loads the selected controller(s). After the design is complete, export the controller(s) from the design tool to the MATLAB workspace.
Use your knowledge of the process to identify the operating points and design the controllers for the plant. If you have already designed the controllers in the MATLAB workspace, import the MPC object to the Multiple MPC Controllers block. You can also use the Simulink Control Design software to compute the operating points, obtain linear models and design the controllers. To learn more, see the Simulink Control Design documentation.

After you add the controllers to the Multiple MPC Controllers block, determine the switching mechanism for the controllers. For example, you can choose one or more measurements from the process to determine when a specific controller is active. This switching signal must round to an integer 1 when the first controller is to be used, to 2 when the second is to be used, and so on. The MMPC block automatically rounds the switching signal to the nearest integer. If the switch signal is less than or equal to zero, or greater than \( N_c \), none of the controllers activates and the block output goes to zero. In general, you will need to use measurements and plant-specific information in order to decide on the controller to be used at a given moment. Thus, generation of the switching signal will usually require special provisions in your Simulink model.

To allow the control system to switch between controllers based on the switching signal when a simulation is running, connect the switching signal to the block’s switch input.

The inactive controllers automatically receive the current manipulated variable and measured output signals so they can update their state estimates. This minimizes bumps in controller transitions (see “Bumpless Transfer in MPC” on page 4-41). It is good practice to select **Enable input port for externally supplied manipulated variables to plant** and connect the actual plant input signal to the \texttt{ext.mv} block input, in which case the block feeds this signal to all its controllers.
Multiple MPC Controller Block Mask Default Settings

The other block mask settings shown in Multiple MPC Controller Block Mask Default Settings on page 3-10 have the same purpose as in the standard MPC Controller block. See “MPC Controller Block” on page 3-3 for details.

See also the mpcswitching and mpcstr demos for example uses of the MMPC block.
Case-Study Examples

- “Introduction” on page 4-2
- “Servomechanism Controller” on page 4-3
- “Paper Machine Process Control” on page 4-28
- “Bumpless Transfer in MPC” on page 4-41
- “Nonlinear Control Using Multiple Models” on page 4-48
- “Using the Tuning Advisor” on page 4-56
- “Bibliography” on page 4-61
**Introduction**

This chapter describes some typical model predictive control applications. Familiarity with LTI models (from the Control System Toolbox product) and Simulink block diagrams will make the examples easier to understand, but you can skip the modeling details to focus on the control aspects.

The first example designs a servomechanism controller. The specifications require a fast servo response despite constraints on a plant input and a plant output.

The second example controls a paper machine headbox. The process is nonlinear, and has three outputs, two manipulated inputs, and two disturbance inputs, one of which is measured for feedforward control.

The third example demonstrates Model Predictive Control Toolbox 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.

The fourth example is a nonlinear chemical reactor (CSTR). It has three inputs and two outputs. 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. 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.
Servomechanism Controller

Position Servomechanism Schematic

System Model

A position servomechanism consists of a DC motor, gearbox, elastic shaft, and a load (see Position Servomechanism Schematic on page 4-3). The differential equations representing this system are

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

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

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 4-4 for more information on these).

If you define the state variables as $x_p = [\theta_L \quad \omega_L \quad \theta_M \quad \omega_M]^T$, then you can convert the above model to an LTI state-space form:
Parameters Used in the Servomechanism Model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value (SI Units)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_\theta$</td>
<td>1280.2</td>
<td>Torsional rigidity</td>
</tr>
<tr>
<td>$k_T$</td>
<td>10</td>
<td>Motor constant</td>
</tr>
<tr>
<td>$J_M$</td>
<td>0.5</td>
<td>Motor inertia</td>
</tr>
<tr>
<td>$J_L$</td>
<td>50$J_M$</td>
<td>Load inertia</td>
</tr>
<tr>
<td>$\rho$</td>
<td>20</td>
<td>Gear ratio</td>
</tr>
<tr>
<td>$\beta_M$</td>
<td>0.1</td>
<td>Motor viscous friction coefficient</td>
</tr>
<tr>
<td>$\beta_L$</td>
<td>25</td>
<td>Load viscous friction coefficient</td>
</tr>
<tr>
<td>$R$</td>
<td>20</td>
<td>Armature resistance</td>
</tr>
</tbody>
</table>

Control Objectives and Constraints

The controller must set the load’s angular position, $\theta_L$, at a desired value by adjusting the applied voltage, $V$. The only measurement available for feedback is $\theta_L$.

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, \( \theta_L \), and one unmeasured, \( T \).

**Defining the Plant Model**

The first step in a design is to define the plant model. The following commands are from the MPC demos file mpcmotormodel.m, which you can run instead of entering the commands manually.

```matlab
% DC-motor with elastic shaft
%
% Parameters (MKS)
%-----------------------------------------------------------
Lshaft=1.0; % Shaft length
dshaft=0.02; % Shaft diameter
shafrho=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=0*Ip/Lshaft; % Torsional rigidity (Torque/angle)
Vshaft=pi*(dshaft^2)/4*Lshaft; % Shaft volume
Mshaft=shafrho*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
```
\[ V_{\text{min}} = -V_{\text{max}}; \]

\%Input/State/Output continuous time form
\%----------------------------------------------------------
\%----------------------------------------------------------
\[
AA = \begin{bmatrix}
0 & 1 & 0 & 0 \\
-Kth/JL & -B\text{load}/JL & Kth/(\text{gear*JL}) & 0 \\
0 & 0 & 0 & 1 \\
Kth/(JM*\text{gear}) & 0 & -Kth/(JM*\text{gear}^2) & -(B\text{motor+Kt}^2/R)/JM
\end{bmatrix};
\]

\[
BB = \begin{bmatrix}
0 \\
0 \\
0 \\
Kt/(R*JM)
\end{bmatrix};
\]

\[
Hyd = \begin{bmatrix}
1 \\
0 \\
0 \\
0
\end{bmatrix};
\]

\[
Hvd = \begin{bmatrix}
Kth \\
0 \\
-Kth/\text{gear} \\
0
\end{bmatrix};
\]

\[
Dyd = 0;
\]

\[
Dvd = 0;
\]

\% 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 4-3. 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 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.

![Design Tool After Specifying Signal Properties](image)

**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 **MPCdesign** and selects it, as in Design Tool After Importing the Plant Model on page 4-8.

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 4-8). 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 4-11). Leave the other values at their defaults for now.
Specifying Constraints
Next, click the Constraints tab. The view shown in Controller Design View, Constraints Pane on page 4-12 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 4-12).

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 4-8). Select Scenario1.

A scenario is a set of simulation conditions. As shown in Simulation Settings View for “Scenario1” on page 4-15, 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 4-15, 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 View for “Scenario1”](image)

**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 4-16. 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 4-16 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 (not shown) 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 4-18 and Manipulated Variable Adjustments on page 4-19.
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 4-18) and the applied voltage stays within its limits (see Manipulated Variable Adjustments on page 4-19).
Manipulated Variable Adjustments

Modifying the Scenario
Finally, increase the step size to $\pi$ radians (select the Scenario1 node and edit the tabular value).

As shown in Servo Response for Step Increase of $\pi$ Radians on page 4-20 and Voltage Adjustments on page 4-20, 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 4-20).
Case-Study Examples

Servo Response for Step Increase of \( \text{Radians} \)

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 4-21 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 4-5. Consider the following command sequence:

```matlab
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 4-14 (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`:

```matlab
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:

```matlab
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:

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

You can also set multiple properties simultaneously.

Set the following properties before continuing with this example:

```matlab
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:

```matlab
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:

```matlab
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,

```matlab
subplot(311)
plot(t, y(:,1), [0 t(end)], pi*[1 1])
```
produces the custom plot shown in Plotting the Output of the sim Command on page 4-24. 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.
Using MPC Tools in Simulink

Block Diagram for the Servomechanism Example on page 4-25 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 4-5.

- MPC Controller is from the MPC Blocks library. Model Predictive Control Toolbox™ Simulink® Block Dialog Box on page 4-26 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 4-22.

Block Diagram for the Servomechanism Example
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).

Servomechanism Simulation Scopes on page 4-27 shows the scope displays for a 20-second simulation. 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.

Servomechanism Simulation Scopes
**Paper Machine Process Control**

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 4-28.

---

**Schematic of Paper Machine Headbox Elements**

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_pmmode1.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 \( z, N_1, N_2 \).

**Linearizing the Nonlinear Model**

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

\[
A = \begin{bmatrix}
-1.9300 & 0 & 0 & 0 \\
0.3940 & -0.4260 & 0 & 0 \\
0 & 0 & -0.6300 & 0 \\
0.8200 & -0.7840 & 0.4130 & -0.4260 \\
\end{bmatrix}; \\
B = \begin{bmatrix}
1.2740 & 1.2740 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1.3400 & -0.6500 & 0.2030 & 0.4060 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}; \\
C = \begin{bmatrix}
0 & 1.0000 & 0 & 0 \\
0 & 0 & 1.0000 & 0 \\
0 & 0 & 0 & 1.0000 \\
\end{bmatrix}; \\
D = \text{zeros}(3,4);
\]

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

```matlab
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:

```matlab
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Units</th>
<th>Nominal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_p$</td>
<td>Manipulated</td>
<td>Feed flow rate</td>
<td>kg/h</td>
<td>0</td>
</tr>
<tr>
<td>$G_w$</td>
<td>Manipulated</td>
<td>White water flow rate</td>
<td>kg/h</td>
<td>0</td>
</tr>
<tr>
<td>$N_p$</td>
<td>Meas. dist.</td>
<td>Feed consistency</td>
<td>%</td>
<td>0</td>
</tr>
<tr>
<td>$N_w$</td>
<td>Unmeas. dist.</td>
<td>White water consistency</td>
<td>%</td>
<td>0</td>
</tr>
</tbody>
</table>

### Output Signal Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Units</th>
<th>Nominal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{-2}$</td>
<td>Measured</td>
<td>Headbox level</td>
<td>m</td>
<td>0</td>
</tr>
<tr>
<td>$N_{-1}$</td>
<td>Measured</td>
<td>Feed tank consistency</td>
<td>%</td>
<td>0</td>
</tr>
<tr>
<td>$N_{-2}$</td>
<td>Measured</td>
<td>Headbox consistency</td>
<td>%</td>
<td>0</td>
</tr>
</tbody>
</table>

### Signal Properties for the Paper Machine Application

#### Initial Controller Design

If necessary, review “Specifying Controller Properties” on page 4-10. 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 $N_j$, 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 4-33.

**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
\( (H_2 \text{ and } N_2) \) 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.

**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:

```
PaperMachine
```
Paper Machine Headbox Control Using MPC Tools in Simulink®

Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-38 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_pmmmodel.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 4-38.
Test, Output Variables on page 4-40 shows the scope display from the “Outputs” block for the setup of Paper Machine Headbox Control Using MPC Tools in Simulink® on page 4-38, 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 4-38) 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 4-40 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.
Bumpless Transfer in MPC

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.

A Model Predictive Control Toolbox demo illustrates this behavior. The following figure shows the block diagram.

**Simulink® Block Diagram for the MPC Bumpless Transfer Demo**

The plant is a stable single-input single-output system. Open-Loop Unit Step Response on page 4-42 shows its open-loop unit step response.
Open-Loop Unit Step Response

MPC Block Configuration Settings on page 4-43 shows the MPC block configuration settings for this case. The demo creates the MPC1 controller object. Its sampling period is 0.5. For additional design details, see the demo documentation.

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 Demo on page 4-41 shows how this is connected).

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 Demo on page 4-41).
MPC Block Configuration Settings

The demo 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 Demo on page 4-41 (unless the plant input saturates at –1 or 1). The controller also monitors the plant output at all times. Thus, the controller can update its estimate of the plant state even when in manual.

The demo also employs the optimization switching option. When the system switches to manual, a nonzero signal enters the controller’s QP Switch input, 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 input 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 Disconnected on page 4-46. The external manipulated variable feedback has been disconnected. 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.

Also, a constant zero value is being fed to the controller’s QP Switch input, which keeps the controller active at all times.
External Manipulated Variable Feedback Disconnected

As shown in Output Response with Manipulated Variable Feedback Disconnected and QP Switching Set to Zero on page 4-47 and Manipulated Variable Adjustments with Manipulated Variable Feedback Disconnected and QP Switching Set to Zero on page 4-47, 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 Disconnected and QP Switching Set to Zero

Manipulated Variable Adjustments with Manipulated Variable Feedback Disconnected and QP Switching Set to Zero
Nonlinear Control Using Multiple Models

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. This example illustrates an alternative: coordination of multiple MPC controllers.

The subject process is a constant-volume continuous stirred-tank reactor (CSTR). The model consists of two nonlinear ordinary differential equations (see [2]). 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/m³
- 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/m³ or greater, Switch 1 sends the constant 1 to its output. Otherwise it sends the constant 2.
- If the reactor concentration is 3 kmol/m³ or greater, Switch 2 passes through the signal coming from Switch 1 (either 1 or 2). Otherwise it 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

```matlab
[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);
```
PlantInputGroup.MV = 3;
PlantInputGroup.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 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.

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 input rate tuning weights will have little impact.

<table>
<thead>
<tr>
<th>Input Weights</th>
<th>Name</th>
<th>Performance Weight</th>
<th>Sensitivity</th>
<th>Tuning Direction</th>
<th>Current Tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>0</td>
<td>-6.3326e-05</td>
<td>Increase</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>In2</td>
<td>0</td>
<td>-3.466e-05</td>
<td>Increase</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>In3</td>
<td>0</td>
<td>-0.000039</td>
<td>Increase</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>In4</td>
<td>0</td>
<td>-0.000116</td>
<td>Increase</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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, further progress will be difficult.
Overall, we have improved the performance by \((26.69 - 20.14)/26.69\) which is more than 20%.
Bibliography


Reference for the Design Tool GUI

This chapter is the reference manual for the Model Predictive Control Toolbox design tool (graphical user interface). For example design tool applications, see the Model Predictive Control Toolbox “Getting Started” or Chapter 4, “Case-Study Examples”.

- “Opening the MPC Design Tool” on page 5-2
- “Menu Bar” on page 5-4
- “Toolbar” on page 5-7
- “Tree View” on page 5-8
- “Importing a Plant Model” on page 5-10
- “Importing a Controller” on page 5-16
- “Exporting a Controller” on page 5-19
- “Signal Definition View” on page 5-21
- “Plant Models View” on page 5-26
- “Controllers View” on page 5-29
- “Simulation Scenarios List” on page 5-33
- “Controller Specifications View” on page 5-36
- “Simulation Scenario View” on page 5-59
- “Tuning Advisor” on page 5-67
- “Response Plots” on page 5-75
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 **MPCdesign**.

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.
Opening the MPC Design Tool

The MPC Controller block lets you design, simulate, and tune model predictive controllers. You can use the MPC Design Tool to create a new controller or modify an existing one. Reference and measured disturbance signals, by default, are external inputs to the MPC block. Alternatively, you can specify custom workspace structures, generated for example by a To Workspace block (see "From Workspace" block for structure format). If the Lock Ahead option is selected, the MPC controller will use future values of the corresponding signal when computing current control actions.

Parameters:
- MPC controller
- Initial controller state

Input signals:
- Use custom
- Reference signal
- Measured disturbance

- Enable input port for measured disturbance
- Enable input port for externally supplied manipulated variables to plant
- Enable input port for input and output limits
- Enable input port for switching optimization off

[OK] [Cancel] [Help] [Apply]
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 5-8). 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

New Design
Creates a new (empty) Model Predictive Control Toolbox design project within the Control and Estimation Tools Manager and assigns it a default name. You can also create a new design using the toolbar (see “Toolbar” on page 5-7).

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 5-7).
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 5-7).

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
You have the following options:

- **Plant model** – Import a plant model using the model import dialog box (see “Importing a Plant Model” on page 5-10).
- **Controller** – Import a controller using the controller import dialog box (see “Importing a Controller” on page 5-16).
Export
Export a controller using the export dialog box (see “Exporting a Controller” on page 5-19). 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 5-8). You can select this option from the keyboard by pressing Ctrl+R, or using the toolbar icon (see “Toolbar” on page 5-7).

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:

- Create a new design
- Load a saved design
- Simulate a current scenario
- Save the current design
- Toggle the output area

For more information on the first four tasks, see the following:

- “New Design” on page 5-4
- “Load” on page 5-4
- “Save” on page 5-5
- “Simulate” on page 5-6

You can identify each tool of the toolbar by placing your mouse over it until an identifying tool tip appears.

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 5-21.
- Plant models list – See “Plant Models View” on page 5-26.
- Controllers list – See “Controllers View” on page 5-29.
- Controller specifications – See “Simulation Scenarios List” on page 5-33.
- Scenarios list – See “Simulation Scenario View” on page 5-59.
- Scenario specifications – See “Controller Specifications View” on page 5-36.

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 5-8), 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 5-16 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 5-16
- “Import to” on page 5-18
- “Buttons” on page 5-18
- “Importing a Linearized Plant Model” on page 5-13
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 MATLAB path, 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 5-8, and “Plant Models View” on page 5-26.)
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 5-10).

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 the Getting Started for an example involving a nonlinear chemical reactor (CSTR).
- 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 the Getting Started 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 5-8), 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 5-16
- “Import to” on page 5-18
- “Buttons” on page 5-18

### 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 5-8.)

The imported controller contains a plant model, which appears in the **Plant models** list. (See “Plant Models View” on page 5-26.)

#### 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” on page 5-19
- “Buttons” on page 5-20

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 5-8). 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” on page 5-22
- “Buttons” on page 5-22
- “Signal Properties Tables” on page 5-22
- “Right-Click Menu Options” on page 5-24
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 5-10).

Import Controller
Clicking this opens the MPC Controller Importer dialog box (see “Importing a Controller” on page 5-16).

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).
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 5-10).
• **Import Controller** – Opens the MPC Controller Importer dialog box (see “Importing a Controller” on page 5-16).

• **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 displays this view (see example below).

![Plant Models View Example](image)

The following sections describe the view's main features:

- “Plant Models List” on page 5-27
- “Model Details” on page 5-27
- “Additional Notes” on page 5-28
- “Buttons” on page 5-28
- “Right-Click Options” on page 5-28
Plant Models View

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Sampling Period</th>
<th>Imported on</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>ss</td>
<td>0</td>
<td>10-Jun-2005 12:13:47</td>
</tr>
<tr>
<td>DCp</td>
<td>ss</td>
<td>0</td>
<td>10-Jun-2005 12:13:54</td>
</tr>
</tbody>
</table>

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 5-27). 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 5-10).

**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 5-10).
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 5-30
- “Controller Details” on page 5-30
- “Additional Notes” on page 5-31
- “Buttons” on page 5-31
- “Right-Click Options” on page 5-32
Controllers List

This table lists all the controllers in your project. The example below lists two controllers, MPC1 and MPC2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Plant Model</th>
<th>Control Interval</th>
<th>Prediction Horizon</th>
<th>Last Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC1</td>
<td>DC</td>
<td>1.0</td>
<td>15</td>
<td>10-Jun-2006 12:12:47</td>
</tr>
<tr>
<td>MPC2</td>
<td>DC</td>
<td>2</td>
<td>10</td>
<td>10-Jun-2006 12:24:01</td>
</tr>
</tbody>
</table>

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 5-8). 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 5-26.)

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 5-37 for more details).

The **Prediction Horizon** field is editable and must be a positive, finite integer. You can also set it in the Controller Specifications view (see “Model and Horizons Tab” on page 5-37 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 5-30). 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 5-16).

Export
Opens the MPC Controller Exporter dialog box (see “Exporting a Controller” on page 5-19).

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 5-16).

**Export Controller**
Opens the MPC Controller Exporter dialog box (see “Exporting a Controller” on page 5-19).
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 5-34
- “Scenario Details” on page 5-35
- “Additional Notes” on page 5-35
- “Buttons” on page 5-35
- “Right-Click Options” on page 5-35
Scenarios List
This table lists all the scenarios in your project. The example below lists two, Scenario1 and Scenario2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Controller</th>
<th>Plant</th>
<th>Closed Loop</th>
<th>Constrained</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario1</td>
<td>MPC1</td>
<td>DC</td>
<td>✔</td>
<td>✔</td>
<td>10</td>
</tr>
<tr>
<td>Scenario2</td>
<td>MPC2</td>
<td>DC</td>
<td>✔</td>
<td>✔</td>
<td>20</td>
</tr>
</tbody>
</table>

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 5-8). 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 5-29.) You can also set this using the Scenario Specifications view (for more discussion, see “Simulation Scenario View” on page 5-59).

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 5-26.) You can also set this in the scenario specifications (for more discussion, see “Simulation Scenario View” on page 5-59).

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 5-59).

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 5-59).
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 5-59).

**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 5-8). 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 5-37
- “Constraints Tab” on page 5-40
- “Constraint Softening” on page 5-43
- “Weight Tuning Tab” on page 5-46
- “Estimation Tab” on page 5-49
- “Right-Click Menus” on page 5-58
Model and Horizons Tab

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 5-10.)
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 1-5, and “Prediction and Control Horizons” on page 1-8.

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 5-38) 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).

```
Number of moves computed per step: 4
Custom move allocation vector: [1 3 3 8]
```

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 1-14.
**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Max Down Rate</th>
<th>Max Up Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflux Rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steam Rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distillate Purity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottoms Purity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 1-10, 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 5-49 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.

<table>
<thead>
<tr>
<th>Constraints on manipulated variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>reflux rate</td>
</tr>
<tr>
<td>Steam rate</td>
</tr>
</tbody>
</table>

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-22). 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.

<table>
<thead>
<tr>
<th>Constraints on output variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Distillate Purity</td>
</tr>
<tr>
<td>Bottoms Purity</td>
</tr>
</tbody>
</table>

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-23). 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 5-40).

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 5-49 for the required format.
**Input Constraints**

An example input constraint softening specification appears below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Minimum</th>
<th>Min Band</th>
<th>Maximum</th>
<th>Max Band</th>
<th>Max Down</th>
<th>Max Up Rate</th>
<th>Max Up Blk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflux Rate</td>
<td>0</td>
<td>55</td>
<td></td>
<td>-10</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Steam Rate</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Name and Units columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-41). 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.
The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-42). 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.

<table>
<thead>
<tr>
<th>Model and Horizons</th>
<th>Constraints</th>
<th>Weight Tuning</th>
<th>Estimation (Advanced)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>More robust</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faster response</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input weights**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>Weight</th>
<th>Rate Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflux Rate</td>
<td>Moler reflux rate</td>
<td>mol/min</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Steam Rate</td>
<td>Steam heating rate</td>
<td>mol/min</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Output weights**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distillate Purity</td>
<td>Distillate product purity</td>
<td>mol %</td>
<td>1.0</td>
</tr>
<tr>
<td>bottoms Purity</td>
<td>bottoms product purity</td>
<td>mol %</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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 5-49 for the required format.

### Input Weights

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>Weight</th>
<th>Rate Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflux Rate</td>
<td>Mol reflux rate</td>
<td>lb/min</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Steam Rate</td>
<td>Steam heating rate</td>
<td>lb/min</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

The *Name*, *Description*, and *Units* columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-41).

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 5-21. 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

The Name, Description, and Units columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 “Viewing and Altering 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, \(4 \times \text{ones}(3,1)\) 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 1-14 for an introduction, and “State Estimation” on page 2-9 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 5-21. 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 5-21. 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 5-21. 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 5-19).
Simulation Scenario View

This view appears whenever you select one of your scenario specification nodes (see “Tree View” on page 5-8). 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:

- “Model and Horizons Tab” on page 5-37
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 5-37).
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Initial Value</th>
<th>Size</th>
<th>Time</th>
<th>Period</th>
<th>Look Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distillate Purity</td>
<td>mol %</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottoms Purity</td>
<td>mol %</td>
<td>Step</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-64.

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Initial Value</th>
<th>Size</th>
<th>Time</th>
<th>Period</th>
<th>Look Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steam Rate</td>
<td>L/h</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Name and Units columns are display-only. To change them, use the signal definition view. (See “Signal Definition View” on page 5-21. 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 5-64.

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.
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 5-21. 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 5-64.

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Initial Value</th>
<th>Size</th>
<th>Time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed Rate</td>
<td>mol/hr</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distillate Pury</td>
<td>mol %</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottoms Pury</td>
<td>mol %</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reflux Rate</td>
<td>mol/hr</td>
<td>Constant</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Initial Value</th>
<th>Size</th>
<th>Time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distillate Pury</td>
<td>mol %</td>
<td>Step</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottoms Pury</td>
<td>mol %</td>
<td>Ramp</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reflux Rate</td>
<td>mol/hr</td>
<td>Sine</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Steam Rate</td>
<td>mol/hr</td>
<td>Pulse</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Feed Rate</td>
<td>mol/hr</td>
<td>Gaussian</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Constant**

The signal will be held at the specified **Initial Value** for the entire simulation.

\[ y = y_0 \text{ for } t \geq 0. \]
Step
Prior to Time, the signal = Initial Value. At Time, the signal changes step-wise by Size units. Its value thereafter = Initial Value + Size.

\[ y = y_0 \text{ for } 0 \leq t < t_0, \quad \text{where } y_0 = \text{Initial Value}, \quad t_0 = \text{Time} \]

\[ y = y_0 + M \text{ for } t > 0, \quad \text{where } M = \text{Size}. \]

Ramp
Prior to Time, the signal = Initial Value. At Time, the signal begins to vary linearly with slope Size.

\[ y = y_0 \text{ for } 0 \leq t < t_0, \quad \text{where } y_0 = \text{Initial Value}, \quad t_0 = \text{Time} \]

\[ y = y_0 + M(t - t_0) \text{ for } t \geq t_0, \quad \text{where } M = \text{Size}. \]

Sine
Prior to Time, the signal = Initial Value. At Time, the signal begins to vary sinusoidally with amplitude Size and period Period.

\[ y = y_0 \text{ for } 0 \leq t < t_0, \quad \text{where } y_0 = \text{Initial Value}, \quad t_0 = \text{Time} \]

\[ y = y_0 + M\sin[\omega(t - t_0)] \text{ for } t \geq t_0, \quad \text{where } M = \text{Size}, \quad \omega = 2\pi/\text{Period}. \]

Pulse
Prior to Time, the signal = Initial Value. At Time, a square pulse of duration Period and magnitude Size occurs.

\[ y = y_0 \text{ for } 0 \leq t < t_0, \quad \text{where } y_0 = \text{Initial Value}, \quad t_0 = \text{Time} \]

\[ y = y_0 + M \text{ for } t_0 \leq t \leq t_0, \quad \text{where } M = \text{Size}, \quad T = \text{Period} \]

\[ y = y_0 \text{ for } t \geq t_0 + T \]

Gaussian
Prior to Time, the signal = Initial Value. At Time, the signal begins to vary randomly about Initial Value with standard deviation Size.
\[ y = y_0 \text{ for } 0 \leq t < t_0, \text{ where } y_0 = \text{Initial Value}, t_0 = \text{Time} \]
\[ y = y_0 + M\text{randn} \text{ for } t \geq t_0, \text{ where } M = \text{Size}. \]

\text{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 5-7), 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 “Tuning Advisor” on page 5-67.

**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.
Tuning Advisor

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 do 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 5-66). 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 column...
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 5-21 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

\[
J = \sum_{i=1}^{T_{\text{stop}}} \left[ \sum_{j=1}^{n_y} (w_j y_{ij}^y)^2 + \sum_{j=1}^{n_u} \left[ (w_j e_{uij})^2 + (w_j h u_{ij})^2 \right] \right]
\]

where \( T_{\text{stop}} \) is the number of controller sampling intervals in the scenario, \( y_{ij} \) 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 \( i \), \( \Delta u_{ij} \) is the change in manipulated variable \( j \) at time step \( i \) (i.e., \( \Delta u_{ij} = u_{ij} - u_{i-1,j} \)), and \( w^y_j \), \( w^u_j \), and \( w^\Delta u_j \) are non-negative performance weights.

- **IAE (Integral of Absolute Error)**. Similar to the ISE but with squared terms replaced by absolute values

\[
J = \sum_{i=1}^{T_{\text{stop}}} \left( \sum_{j=1}^{n_y} |w^y_j e_{yij}| + \sum_{j=1}^{n_u} (|w^u_j e_{uij}| + |w^\Delta u_j \Delta u_{ij}|) \right)
\]

The IAE gives less emphasis to any large deviations.

- **ITSE (time-weighted Integral of Squared Errors)**

\[
J = \sum_{i=1}^{T_{\text{stop}}} i \Delta t \left( \sum_{j=1}^{n_y} (w^y_j e_{yij})^2 + \sum_{j=1}^{n_u} ((w^u_j e_{uij})^2 + (w^\Delta u_j \Delta u_{ij})^2) \right)
\]

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)**

\[
J = \sum_{i=1}^{T_{\text{stop}}} i \Delta t \left( \sum_{j=1}^{n_y} |w^y_j e_{yij}| + \sum_{j=1}^{n_u} (|w^u_j e_{uij}| + |w^\Delta u_j \Delta u_{ij}|) \right)
\]

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, \( w^y_j \), \( w^u_j \), and \( w^\Delta u_j \). 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 \( w^y_j \) 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 $w_j^{Au}$ 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 “Response Plots” on page 5-75).
Sensitivities and Tuning Advice

Click Analyze its to 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 $w_j$ 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 MATLAB windows 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.
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 5-76
- “Displaying Multiple Scenarios” on page 5-77
- “Viewing Selected Variables” on page 5-78
- “Grouping Variables in a Single Plot” on page 5-79
- “Normalizing Response Amplitudes” on page 5-79
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** – Interpolate 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
Reference for the Design Tool GUI

Marker” on page 5-76). 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.
B
blocking
   specification 5-37
buttons
   toolbar 5-7

C
constraints
   constraint softening 5-43
   hard vs. soft 1-13
   manipulated variables 5-41
   optimization problem 2-5
   output variables 5-42
   overview 1-7
   soft 1-13
   specification dialog 5-40
   tolerance band 1-13
Control and Estimation Tools Manager
   MPC Design Tool 5-2
control horizon 1-8
   specification 5-37
control interval 1-5
   specification 5-37
controller settings
   blocking 5-37
   control interval 5-37
   estimation 5-49
   horizons 5-37
   overall robustness 5-46
   plant model 5-37
controller specifications
   constraint softening 5-43
   constraints 5-40
   relaxation bands 5-43
   weights 5-46
controllers
   copying 5-29
   creating 5-29
   deleting 5-29
   export button 5-29
   export menu 5-6
   exporting controllers 5-19
   from MAT-file 5-17
   from workspace 5-17
   import button 5-29
   import menu 5-5
   importing 5-16
   notes 5-29
   renaming 5-8
   specification 5-36
   summary listing 5-29

D
data markers 5-76
delays
   horizons for 1-8
description
   specification 5-22
design tool
   closing 5-5
   opening 5-2
   renaming tree nodes 5-8
   tree navigation 5-8
disturbance estimation
   controller settings 5-49
disturbances
   inputs 5-53
   outputs 5-51
   simulation specification 5-59
duration
   specification 5-59

E
estimation
   controller settings 5-49
   Kalman filter 2-11
   model form 2-9
Index

state 1-14

**F**

feedforward compensation 1-3
file menu
  close design tool 5-5
  load saved design 5-4
  new design 5-4
  save current design 5-5

**H**

hold
  zero-order 1-7
horizons
  control 1-8
  prediction 1-8

**I**

identified models 2-20
importing
  controller from MAT-file 5-17
  controller from workspace 5-17
  controllers 5-16
  model from MAT-file 5-12
  model from workspace 5-11
inputs
  property specifications 5-22
  inverse-response 1-9

**K**

Kalman filter 2-11

**L**

linearization
  during model import 5-13
  offset 2-4
  operating point selection 5-15
  options 5-15
  procedure 5-14
  load button 5-7

**M**

manipulated variables
  example 1-2
  setpoints 1-12
MAT-file
  controller export 5-19
  controller import 5-17
  model import 5-12
measured disturbances
  example 1-2
menus
  file 5-4
  menu bar 5-4
  MPC 5-5
models
  controller specification 5-37
  defining disturbances 5-49
  deleting 5-26
  disturbance 1-4
  from MAT-file 5-12
  from workspace 5-11
  identified 2-20
  import tool 5-10
  importing 5-26
  importing plant model 5-5
  linearization during import 5-13
  noise 5-49
  notes 5-26
  output disturbance 2-10
  summary list 5-26
  viewing details 5-26
moves
  suppression 1-12
MPC menu
export 5-6
import controller 5-5
import plant model 5-5
simulation start 5-6

N
names
  specification 5-22
new design button 5-7
nodes
  controller specification 5-8
  controllers list 5-8
  MPC project/task 5-8
  plant models list 5-8
  renaming 5-8
  simulation scenario specification 5-8
  simulation scenarios list 5-8
  types 5-8
noise
  controller settings 5-56
nominal conditions
  definition 2-4
nominal values
  specification 5-22
notes
  controllers 5-29
  models 5-26
  scenarios 5-33

O
observer 2-9
offset
  linearization 2-4
optimization 1-7
  setpoint tracking 1-10
outputs
  disturbance model 2-10
  property specifications 5-23

P
plant model importer 5-10
plant/model mismatch
  simulation specification 5-59
plants
  MIMO 1-10
  nonminimum phase 1-9
  nonsquare 1-10
  SISO 1-2
  state 2-3
plots
  data markers 5-76
  normalizing 5-79
  simulation responses 5-75
  variable selection 5-79
prediction horizon 1-8
  specification 5-37
projects
  controller import 5-18
  importing model 5-12
  load 5-4
  new 5-2
  renaming 5-8
  save 5-5
properties
  output signals 5-23
  signal specifications 5-22

R
relaxation bands
  specification 5-43
response plots 5-75 to 5-76
  See also plots
robustness
  global setting 5-46

S
save button 5-7
scenarios
  comparing 5-77
  controller option 5-59
copying 5-33
creating 5-33
current 5-6
deleting 5-33
duration 5-59
notes 5-33
plant model option 5-59
renaming 5-8
settings 5-59
simulation button 5-59
summary list 5-33
setpoints
  specification in simulations 5-59
signal definition
  overview graphic 5-22
  property specification table 5-22
  specification view 5-21
signal properties
  inputs 5-22
  specification 5-22
signal type
  specification 5-22
signals
  description 5-22
  names 5-22
  nominal values 5-22
  types 1-3
  types definition 5-22
  units 5-22
simulate button 5-7
simulations
  closed-loop option 5-59
duration 5-59
open-loop option 5-59
response plots 5-75
start using MPC menu 5-6
state
controller 3-6
observer 2-11
observer form 2-9
state estimation 1-14
states
  plant 2-3

T
  tasks
    importing model 5-12
tolerance bands 1-13
toolbar
    overview 5-7
tree view
    node types 5-8
    overview 5-8
Tuning Advisor 5-67

U
  units
    specification 5-22
unmeasured disturbances
  example 1-2
using identified models, system identification
  model conversion 2-20
Using the Tuning Advisor 4-56

V
  views
    signal definition 5-21
    tree 5-8

W
  weights
    inputs 5-47
    optimization problem 2-5
    outputs 5-48
specification dialog 5-46
tuning 5-46
workspace
controller export 5-19
importing model from 5-11