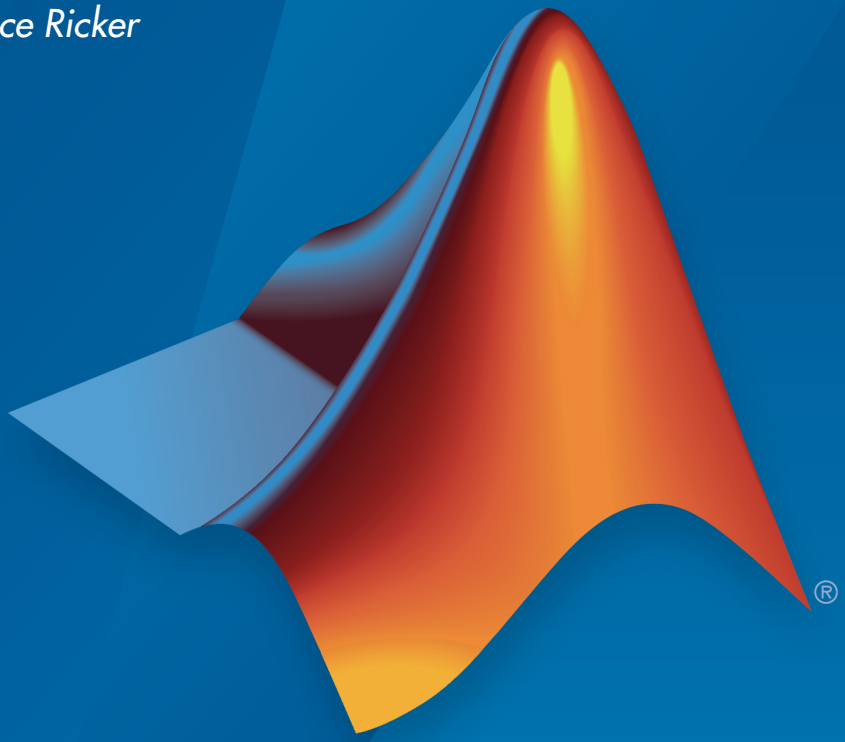
Model Predictive Control Toolbox™

User's Guide

Alberto Bemporad
Manfred Morari
N. Lawrence Ricker



MATLAB®

R2015a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

Model Predictive Control Toolbox™ User's Guide

Revision History

October 2004	First printing	New for Version 2.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 2.2.2 (Release 2006a)
September 2006	Online only	Revised for Version 2.2.3 (Release 2006b)
March 2007	Online only	Revised for Version 2.2.4 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 2.3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.1.1 (Release 2009b)
March 2010	Online only	Revised for Version 3.2 (Release 2010a)
September 2010	Online only	Revised for Version 3.2.1 (Release 2010b)
April 2011	Online only	Revised for Version 3.3 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)
March 2012	Online only	Revised for Version 4.1 (Release 2012a)
September 2012	Online only	Revised for Version 4.1.1 (Release 2012b)
March 2013	Online only	Revised for Version 4.1.2 (Release R2013a)
September 2013	Online only	Revised for Version 4.1.3 (Release R2013b)
March 2014	Online only	Revised for Version 4.2 (Release R2014a)
October 2014	Online only	Revised for Version 5.0 (Release R2014b)
March 2015	Online only	Revised for Version 5.0.1 (Release 2015a)

Contents

1 Introduction

2 Model Predictive Control Problem Setup

4 Case-Study Examples

5 Adaptive MPC Design

6 Explicit MPC Design

7 Gain Scheduling MPC Design

8 Reference for the Design Tool GUI

1

Introduction

Specifying Scale Factors

In this section...

Overview

Recommended practice includes specification of scale factors for each plant input and output variable. This is especially important when certain variables will have much larger or smaller magnitudes than others.

The scale factor should equal (or approximate) the variable's span. *Span* is the difference between its maximum and minimum value in engineering units, i.e., the unit of measure specified in the plant model. Internally, MPC divides each plant input and output signal by its scale factor to generate dimensionless signals.

The potential benefits of scaling are as follows:

- Default MPC tuning weights work best when all signals are of order unity. Appropriate scale factors make the default weights a good starting point for controller tuning and refinement.
- When choosing cost function weights you can focus on the relative priority of each term rather than a combination of priority and signal scale.
- Improved numerical conditioning. When values are scaled, round-off errors have less impact on calculations.

Once you have tuned the controller, changing a scale factor is likely to affect performance and might necessitate retuning. Best practice is to establish scale factors at the beginning of controller design and hold them constant thereafter.

Defining Scale Factors

To identify scale factors, estimate the span of each plant input and output variable in engineering units.

- If the signal has known bounds, use the difference between the upper and lower limit.

- If you do not know the signal bounds, consider running open-loop plant model simulations in which you vary the inputs over their likely ranges and record output signal spans.
- If you have no idea, use the default scale factor (=1).

After you create the MPC controller object using the `mpc` command), set the scale factor property for each plant input and output variable.

For example, the following commands define a random plant, specifies the signal types, and creates and specifies a scale factor for each signal.

```
% Random plant for illustrative purposes: 5 inputs, 3 outputs
Plant = drss(4,3,5);
Plant.InputName = {'MV1','UD1','MV2','UD2','MD'};
Plant.OutputName = {'UO','MO1','MO2'};

% Example signal spans
Uspan = [2, 20, 0.1, 5, 2000];
Yspan = [0.01, 400, 75];

% Example signal type specifications
iMV = [1 3];   iMD = 5;   iUD = [2 4]; iDV = [iMD, iUD];
Plant = setmpcsignals(Plant, 'MV', iMV, 'MD', iMD, 'UD', iUD, ...
    'MO', [2 3], 'UO', 1);
Plant.d(:,iMV) = 0;   % MPC requires zero direct MV feed-through

% Controller object creation.  Ts = 0.3 for illustration.
MPCobj = mpc(Plant, 0.3);

% Override default scale factors using specified spans
for i = 1:2
    MPCobj.MV(i).ScaleFactor = Uspan(iMV(i));
end

% NOTE:  DV sequence is MD followed by UD
for i = 1:3
    MPCobj.DV(i).ScaleFactor = Uspan(iDV(i));
end
for i = 1:3
    MPCobj.OV(i).ScaleFactor = Yspan(i);
end
```

Once you have set the scale factors and have begun to tune controller performance, hold the scale factors constant.

See Also

`mpc`

Related Examples

- Using Scale Factor to Facilitate Weight Tuning

More About

- “Choosing Sample Time and Horizons”

Choosing Sample Time and Horizons

In this section...

Sample Time

Duration

Recommended practice is to choose the control interval duration (controller property `Ts`) initially, and then hold it constant as you tune other controller parameters. If it becomes obvious that the original choice was poor, you can revise `Ts`. If you do so, you might then need to re-tune other settings.

Qualitatively, as `Ts` decreases, rejection of unknown disturbance usually improves and then plateaus. The `Ts` value at which performance plateaus depends on the plant dynamic characteristics.

On the other hand, as `Ts` becomes small, the computational effort increases dramatically. Thus, the optimal choice is a balance of performance and computational effort.

In Model Predictive Control, the prediction horizon, p is also an important consideration. If one chooses to hold the prediction horizon duration (the product p*`Ts`) constant, p must vary inversely with `Ts`. Many array sizes are proportional to p. Thus, as p increases memory requirements increase, as does the time needed to solve each QP.

Consider the following when choosing `Ts`:

- Rough guideline: establish the minimum desired closed-loop response time and set `Ts` between 0.1 and 0.25 of this.
- Run at least one simulation test to see whether unmeasured disturbance rejection improves significantly when `Ts` is halved. If so, consider revising `Ts`.
- For process control, `Ts` >> 1 s is common, especially when MPC supervises lower-level single-loop controllers. Other applications (e.g., automotive, aerospace) can require `Ts` < 1 s. If the time needed to solve the QP in real time exceed the desired control interval, consider the Explicit MPC option.

- Plants with delay: the number of state variables needed to model delays is inversely proportional to `Ts`.
- Open-loop unstable plants: if p*`Ts` is too large, such that the plant step responses become infinite during this amount of time, key parameters needed for MPC calculations become undefined, generating an error message.

Units

The controller inherits its time unit from the plant model. Specifically, the controller uses the `TimeUnit` property of the plant model LTI object. This property defaults to seconds.

Prediction Horizon

Suppose the current control interval is k. The *prediction horizon*, p, is the number of future control intervals the MPC controller must evaluate by prediction when optimizing its MVs at control interval k.

Tips

- Recommended practice is to choose p early in the controller design and then hold it constant while tuning other controller settings, such as the cost function weights. In other words, you should not use p adjustments for controller tuning. Rather, the value of p should be such that the controller is internally stable and anticipates constraint violations early enough to allow corrective action.
- If the desired closed-loop response time is T and the control interval is T_s, try p such that $T \approx pT_s$.
- Plant delays impose a lower bound on the possible closed-loop response times. Choose p accordingly. Use the `review` command to check for a violation of this condition.
- Recommended practice is to increase p until further increases have a minor impact on performance. If the plant is open-loop unstable, the maximum possible p is the number of control intervals required for the plant's open-loop step response to become infinite. $p > 50$ is rarely necessary unless `Ts` is too small.
- Unfavorable plant characteristics combined with a small p can generate an internally unstable controller. Use `review` to check for this condition, and increase p if possible. If p is already large, consider the following.
  - Increase `Ts`.
  - Increase the cost function weights on MV increments.

- Modify the control horizon and/or use MV blocking (see “Manipulated Variable Blocking”).
- Use a small p in combination with terminal weighting to approximate LQR behavior. (See “Terminal Weights and Constraints”.)

Control Horizon

The control horizon, m, is the number of MV moves to be optimized at control interval k. The control horizon falls between 1 and the prediction horizon p. The default is $m = 2$. Regardless of your choice for m, when the controller operates, the optimized MV move at the beginning of the horizon is used and any others are discarded.

Tips

Reasons to keep $m << p$ are as follows:

- Small m means fewer variables to compute in the QP solved at each control interval, which promotes faster computations.
- If the plant includes delays, $m < p$ is essential. Otherwise, some MV moves might not affect any of the plant outputs prior to the end of the prediction horizon, leading to a singular QP Hessian matrix. Use `review` to check for a violation of this condition.
- Small m promotes (but does not guarantee) an internally-stable controller.

More About

- “Specifying Constraints”

Specifying Constraints

In this section...

Input and Output Constraints

By default, when you create a controller object using the `mpc` command, no constraints exist. To include a constraint, set the appropriate controller property. The following table summarizes the controller properties used to define most MPC Toolbox constraints. (MV = plant manipulated variable; OV = plant output variable; MV increment = $u(k) - u(k-1)$.

To include this constraint	Set this controller property	Soften constraint by setting
Lower bound on *i*th MV	`MV(i).Min > -Inf`	`MV(i).MinECR > 0`
Upper bound on *i*th MV	`MV(i).Max < Inf`	`MV(i).MaxECR > 0`
Lower bound on *i*th OV	`OV(i).Min > -Inf`	`OV(i).MinECR > 0`
Upper bound on *i*th OV	`OV(i).Max < Inf`	`OV(i).MaxECR > 0`
Lower bound on *i*th MV increment	`MV(i).RateMin > -Inf`	`MV(i).RateMinECR > 0`
Upper bound on *i*th MV increment	`MV(i).RateMax < Inf`	`MV(i).RateMaxECR > 0`

See "Constraints" for the equations describing the corresponding constraints.

Tips

For MV bounds:

- Include known physical limits on the plant MVs as hard MV bounds.
- Include MV increment bounds when there is a known physical limit on the rate of change, or your application requires you to prevent large increments for some other reason.
- Do not include both hard MV bounds and hard MV increment bounds on the same MV. They might conflict. If both types of bounds are important, soften one.

For OV bounds:

- Do not include OV bounds unless they are essential to your application. As an alternative to setting an OV bound, you can define an OV reference and set its cost function weight to keep the OV close to its reference value.
- All OV constraints should be softened.
- Consider leaving the OV unconstrained for some prediction horizon steps. See "Time-Varying Weights and Constraints".
- Consider a time-varying OV constraint that is easy to satisfy early in the horizon, gradually tapering to a more strict constraint. See "Time-Varying Weights and Constraints".
- Do not include OV constraints that are impossible to satisfy. Even if soft, such constraints can cause unexpected controller behavior. For example, consider a SISO plant with five sampling periods of delay. An OV constraint prior to the sixth prediction horizon step is, in general, impossible to satisfy. You can use the `review` command to check for such impossible constraints, and use a time-varying OV bound instead. See "Time-Varying Weights and Constraints".

Constraint Softening

Hard constraints are constraints that must be satisfied by the quadratic programming (QP) solution. If it is mathematically impossible to satisfy a hard constraint at a given control interval, k, the QP is *infeasible*. In this case, the controller returns an error status, and sets the manipulated variables (MVs) to $u(k) = u(k–1)$, i.e., no change. If the condition leading to infeasibility is not resolved, infeasibility can continue indefinitely, leading to a loss of control.

Disturbances and prediction errors are inevitable in practice. Thus, a constraint violation could occur in the plant even though the controller predicts otherwise. A feasible QP solution does not guarantee that all hard constraints will be satisfied when the optimal MV is used in the plant.

If the only constraints in your application are bounds on MVs, the MV bounds can be hard constraints, as they are by default. MV bounds alone cannot cause infeasibility. The same is true when the only constraints are on MV increments.

On the other hand, a hard MV bound in combination with a hard MV increment constraint can lead to infeasibility. For example, an upset or operation under manual control could cause the actual MV used in the plant to exceed the specified bound during

interval k–1. If the controller is in automatic during interval k, it must return the MV to a value within the hard bound. If the MV exceeds the bound by too much, the hard increment constraint might make correcting the bound violation in the next interval impossible.

When there are hard constraints on plant outputs or hard custom constraints (on linear combinations of plant inputs and outputs) and the plant is subject to disturbances, QP infeasibility is a distinct possibility.

All MPC toolbox constraints (except slack variable nonnegativity) can be *soft*. When a constraint is soft, the controller may deem an MV optimal even though it predicts a violation of that constraint. If all plant output, MV increment, and custom constraints are soft (as they are by default), QP infeasibility does not occur. However, controller performance might be sub-standard.

To soften a constraint, set the corresponding ECR value to a positive value (zero implies a hard constraint). The larger the ECR value, the more likely the controller will deem it optimal to violate the constraint in order to satisfy your other performance goals. The Model Predictive Control Toolbox™ software provides default ECR values but, as for the cost function weights, you might need to tune the ECR values in order to achieve acceptable performance.

To better understand how constraint softening works, suppose your cost function uses $w^u_{i,j} = w^{\Delta u}_{i,j} = 0$, giving both the MV and MV increments zero weight in the cost function. Only the output reference tracking and constraint violation terms are nonzero. In this case, the cost function is:

$$J(z_k) = \sum_{j=1}^{n_y}\sum_{i=1}^{p}\left\{\frac{w^y_{i,j}}{s^y_j}\left[r_j(k+i|k) - y_j(k+i|k)\right]\right\}^2 + \rho\ \varepsilon_k^2.$$

Suppose you have also specified hard MV bounds with $V^u_{j,min}(i) = 0$ and $V^u_{j,max}(i) = 0$. Then these constraints simplify to:

$$\frac{u_{j,min}(i)}{s^u_j} \le \frac{u_j(k+i-1|k)}{s^u_j} \le \frac{u_{j,max}(i)}{s^u_j},\ i = 1:p,\quad j = 1:n_u.$$

Thus, the slack variable, ϵ_k, no longer appears in the above equations. You have also specified soft constraints on plant outputs with $V^y_{j,min}(i) > 0$ and $V^y_{j,max}(i) > 0$.

$$\frac{y_{j,min}(i)}{s^y_j} - {}_kV^y_{j,min}(i) \le \frac{y_j(k+i|k)}{s^y_j} \le \frac{y_{j,max}(i)}{s^y_j} + {}_kV^y_{j,max}(i),\ i = 1:p, \quad j = 1:n_y.$$

Now suppose a disturbance has pushed a plant output above its specified upper bound, but the QP with hard output constraints would be feasible, i.e., all constraint violations could be avoided in the QP solution. The QP involves a trade-off between output reference tracking and constraint violation. The slack variable, ϵ_k, must be nonnegative. Its appearance in the cost function discourages, but does not prevent, an optimal $\epsilon_k > 0$. A larger ρ_ϵ weight, however, increases the likelihood that the optimal ϵ_k will be small or zero.

If the optimal $\epsilon_k > 0$, at least one of the bound inequalities must be active (at equality). A relatively large $V^y_{j,max}(i)$ makes it easier to satisfy the constraint with a small ${}_\epsilon k$. In that case,

$$\frac{y_j(k+i|k)}{s^y_j}$$

can be larger, without exceeding

$$\frac{y_{j,max}(i)}{s^y_j} + {}_kV^y_{j,max}(i).$$

Notice that $V^y_{j,max}(i)$ does not set an upper limit on the constraint violation. Rather, it is a tuning factor determining whether a soft constraint is easy or difficult to satisfy.

Tips

- Use of dimensionless variables simplifies constraint tuning. Define appropriate scale factors for each plant input and output variable. See “Specifying Scale Factors” on page 1-2.

- Use the ECR parameter associated with each constraint (see above table) to indicate the relative magnitude of a tolerable violation. Rough guidelines are as follows:
  - 0: no violation allowed (hard constraint)
  - 0.05: very small violation allowed (nearly hard)
  - 0.2: small violation allowed (quite hard)
  - 1: average softness
  - 5: greater-than-average violation allowed (quite soft)
  - 20: large violation allowed (very soft)
- Use the controller's overall constraint softening parameter (controller object property: `Weights.ECR`) to penalize a tolerable soft constraint violation relative to the other cost function terms. Set the `Weights.ECR` property such that the corresponding penalty is 1 to 2 orders of magnitude greater than the typical sum of the other three cost function terms. If constraint violations seem too large during simulation tests, try increasing `Weights.ECR` by a factor of 2 to 5.

  Be aware, however, that an excessively large `Weights.ECR` distorts MV optimization, leading to inappropriate MV adjustments when constraint violations occur. To check for this, display the cost function value during simulations. If its magnitude increases by more than 2 orders of magnitude when a constraint violation occurs, consider decreasing `Weights.ECR`.
- Disturbances and prediction errors will lead to unexpected constraint violations in a real system. Attempting to prevent this by making constraints harder often degrades controller performance.

See Also

`review`

More About

- "Time-Varying Weights and Constraints"
- "Terminal Weights and Constraints"
- "Optimization Problem"

Tuning Weights

In this section...

A Model Predictive Controller design usually requires some tuning of the cost function weights. This topic provides tuning tips. See "Optimization Problem" for details on the cost function equations.

Initial Tuning

- Prior to tuning the cost function weights, specify scale factors for each plant input and output variable "Specifying Scale Factors" on page 1-2. Hold these scale factors constant as you tune the controller.
- At any point during tuning, use the `sensitivity` and `review` commands to obtain diagnostic feedback. The `sensitivity` command is specifically intended to help with cost function weight selection.
- Change a weight by setting the appropriate controller property, as follows:

To change this weight	Set this controller property	Array size
OV reference tracking (w^y)	`Weights.OV`	p-by-n_y
MV reference tracking (w^u)	`Weights.MV`	p-by-n_u
MV increment suppression ($w^{\Delta u}$)	`Weights.MVRate`	p-by-n_u

Here, MV is a plant manipulated variable, and n_u is the number of MVs. OV is a plant output variable, and n_y is the number of OVs. Finally,p is the number of steps in the prediction horizon.

If a weight array contains $n < p$ rows, the controller duplicates the last row to obtain a full array of p rows. The default ($n = 1$) minimizes the number of parameters to be tuned, and is therefore recommended. See "Time-Varying Weights and Constraints" for an alternative.

Tips for Setting OV Weights

- Considering the n_y OVs, suppose that n_{yc} must be held at or near a reference value (setpoint). If the ith OV is not in this group, set `Weights.OV(:,i)` = 0.
- If $n_u \geq n_{yc}$, it is usually possible to achieve zero OV tracking error at steady state, provided that at least n_{yc} MVs are not at a bound. The default `Weights.OV = ones(1,ny)` is a good starting point in this case.

  If $n_u > n_{yc}$, however, you have excess degrees of freedom. Unless you take preventive measures, therefore, the MVs may drift even when the OVs are near their reference values.

  - The most common preventive measure is to define reference values (targets) for the number of excess MVs you have, $n_u - n_{yc}$ MVs. Such targets can represent economically or technically desirable steady-state values.
  - An alternative measure is to set $w_{\Delta u} > 0$ for at least $n_u - n_{yc}$ MVs to discourage the controller from changing them.
- If $n_u < n_{yc}$, you do not have enough degrees of freedom to keep all required OVs at a setpoint. In this case, consider prioritizing reference tracking. To do so, set `Weights.OV(:,i) > 0` to specify the priority for the ith OV. Rough guidelines for this are as follows:

  - 0.05 Low priority: large tracking error acceptable
  - 0.2 Below-average priority
  - 1 Average priority – the default. Use this if $n_{yc} = 1$.
  - 5 Above average priority
  - 20 High priority: small tracking error desired

Tips for Setting MV Weights

By default, `Weights.MV = zeros(1,nu)`. If some MVs have targets, the corresponding MV reference tracking weights must be nonzero. Otherwise, the targets are ignored. If the number of MV targets is less than $(n_u - n_{yc})$, try using the same weight for each. A suggested value is 0.2, the same as below-average OV tracking. This value allows the MVs to move away from their targets temporarily to improve OV tracking.

Otherwise, the MV and OV reference tracking goals are likely to conflict. Prioritize by setting the `Weights.MV(:,i)` values in a manner similar to that suggested for

`Weights.OV` (see above). Typical practice sets the average MV tracking priority lower than the average OV tracking priority (e.g., 0.2 < 1).

If the *i*th MV does not have a target, set `Weights.MV(:,i)` = 0 (the default).

Tips for Setting MVRate Weights

- By default, `Weights.MVRate = 0.1*ones(1,nu)`. The reasons for this default include:
  - If the plant is open-loop stable, large increments are unnecessary and probably undesirable. For example, when model predictions are imperfect, as is always the case in practice, more conservative (smaller) increments usually provide more robust controller performance, although also but poorer reference tracking.
  - These values force the QP Hessian matrix to be positive-definite, such that the QP has a unique solution if no constraints are active.

  To encourage the controller to use even smaller increments for the *i*th MV, increase the `Weights.MVRate(:,i)` value.
- If the plant is open-loop unstable, you might need to decrease the average `Weight.MVRate` value to allow sufficiently rapid response to upsets.

Tips for Setting ECR Weights

See "Constraint Softening" on page 1-9 for tips regarding the `Weights.ECR` property.

Testing and Refinement

To focus on tuning individual cost function weights, perform closed-loop simulation tests under the following conditions:

- No constraints.
- No prediction error. The controller's prediction model should be identical to the plant model.

  Both the design tool and the `sim` function provide the option to simulate under these conditions.

Use changes in the reference and measured disturbance signals (if any) to force a dynamic response. Based on the results of each test, consider changing the magnitudes of selected weights.

One suggested approach is to use constant `Weights.OV(:,i) = 1` to signify "average OV tracking priority," and adjust all other weights to be relative to this value. Use the `sensitivity` command for guidance. Use the `review` command to check for typical tuning issues, such as lack of closed-loop stability.

See "Adjusting Disturbance and Noise Models" for tests focusing on the controller's ability to reject arbitrary disturbances.

Robustness

Once you have weights that work well under the above conditions, check for sensitivity to prediction error. There are several ways to do so:

- If you have a nonlinear plant model of your system, such as a Simulink® model, simulate the closed-loop performance at operating points other than that for which the LTI prediction model applies.
- Alternatively, run closed-loop simulations in which the LTI model representing the plant differs (such as in structure or parameter values) from that used at the MPC prediction model. Both the design tool and the `sim` function provide the option to simulate under these conditions.

If controller performance seems to degrade significantly in comparison to tests with no prediction error, for an open-loop stable plant, consider making the controller less aggressive. In the design tool, you can do so using the performance/robustness trade-off slider adjustment. At the command line, you can make the following changes to increase controller aggressiveness:

- Increase all `Weight.MVRate` values by a multiplicative factor of order 2.
- Decrease all `Weight.OV` and `Weight.MV` values by dividing by the same factor.

After making these adjustments, reevaluate performance with and without prediction error.

- If both are now acceptable, stop tuning the weights
- If there is improvement but still too much degradation with model error, repeat the above weight adjustments.
- If the change does noticeably improve performance, restore the original weights and focus on state estimator tuning (see "Adjusting Disturbance and Noise Models".

Finally, if tuning changes do not provide adequate robustness, consider one of the following options:

- Adaptive MPC control
- Gain-scheduled MPC control

Related Examples

- Tuning Controller Weights
- “Setting Targets for Manipulated Variables”

More About

- “Optimization Problem”
- “Specifying Constraints” on page 1-8
- “Adjusting Disturbance and Noise Models”

2

Model Predictive Control Problem Setup

Optimization Problem

In this section...

Overview

Model Predictive Control solves an optimization problem – specifically, a quadratic program (QP) – at each control interval. The solution determines the manipulated variables (MVs) to be used in the plant until the next control interval.

This QP problem includes the following features:

- The objective, or "cost", function — A scalar, nonnegative measure of controller performance to be minimized.
- Constraints — Conditions the solution must satisfy, such as physical bounds on MVs and plant output variables.
- Decision — The MV adjustments that minimizes the cost function while satisfying the constraints.

The following sections describe these features in more detail.

Standard Cost Function

The standard cost function is the sum of four terms, each focusing on a particular aspect of controller performance, as follows:

$$J(z_k) = J_y(z_k) + J_u(z_k) + J_{\Delta u}(z_k) + J\ (z_k).$$

Here, z_k is the QP decision. As described below, each term includes weights that help you balance competing objectives. MPC controller provides default weights but you will usually need to adjust them to tune the controller for your application.

Output Reference Tracking

In most applications, the controller must keep selected plant outputs at or near specified reference values. MPC controller uses the following scalar performance measure:

$$J_y(z_k) = \sum_{j=1}^{n_y} \sum_{i=1}^{p} \left\{ \frac{w_{i,j}^y}{s_j^y} \left[r_j(k+i|k) - y_j(k+i|k) \right] \right\}^2 .$$

Here,

- k — Current control interval.
- p — Prediction horizon (number of intervals).
- n_y — Number of plant output variables.
- z_k — QP decision, given by:

$$z_k^T = \left[u(k|k)^T \quad u(k+1|k)^T \quad \cdots \quad u(k+p-1|k)^T \quad {}_k \right].$$

- $y_j(k+i|k)$ — Predicted value of jth plant output at ith prediction horizon step, in engineering units.
- $r_j(k+i|k)$ — Reference value for jth plant output at ith prediction horizon step, in engineering units.
- s_j^y — Scale factor for jth plant output, in engineering units.
- $w_{i,j}^y$ — Tuning weight for jth plant output at ith prediction horizon step (dimensionless).

The values n_y, p, s_j^y, and $w_{i,j}^y$ are controller specifications, and are constant. The controller receives $r_j(k+i|k)$ values for the entire prediction horizon. The controller uses the state observer to predict the plant outputs. At interval k, the controller state estimates and MD values are available. Thus, J_y is a function of z_k only.

Manipulated Variable Tracking

In some applications, i.e. when there are more manipulated variables than plant outputs, the controller must keep selected manipulated variables (MVs) at or near specified target values. MPC controller uses the following scalar performance measure:

$$J_u(z_k) = \sum_{j=1}^{n_u} \sum_{i=0}^{p-1} \left\{ \frac{w_{i,j}^u}{s_j^u} \left[u_j(k+i|k) - u_{j,target}(k+i|k) \right] \right\}^2 .$$

Here,

- k — Current control interval.
- p — Prediction horizon (number of intervals).
- n_u — Number of manipulated variables.
- z_k — QP decision, given by:

$$z_k^T = \left[u(k|k)^T \quad u(k+1|k)^T \quad \cdots \quad u(k+p-1|k)^T \quad _k \right].$$

- $u_{j,target}(k+i|k)$ — Target value for jth MV at ith prediction horizon step, in engineering units.
- s_j^u — Scale factor for jth MV, in engineering units.
- $w_{i,j}^u$ — Tuning weight for jth MV at ith prediction horizon step (dimensionless).

The values n_u, p, s_j^u, and $w_{i,j}^u$ are controller specifications, and are constant. The controller receives $u_{j,target}(k+i|k)$ values for the entire horizon. The controller uses the state observer to predict the plant outputs. Thus, J_u is a function of z_k only.

Manipulated Variable Move Suppression

Most applications prefer small MV adjustments (*moves*). MPC uses the following scalar performance measure:

$$J_{\Delta u}(z_k) = \sum_{j=1}^{n_u} \sum_{i=0}^{p-1} \left\{ \frac{w_{i,j}^{\Delta u}}{s_j^u} \left[u_j(k+i|k) - u_j(k+i-1|k) \right] \right\}^2 .$$

Here,

Here,

- k — Current control interval.
- p — Prediction horizon (number of intervals).
- n_u — Number of manipulated variables.
- z_k — QP decision, given by:

$$z_k^T = \begin{bmatrix} u(k \mid k)^T & u(k+1 \mid k)^T & \cdots & u(k+p-1 \mid k)^T \end{bmatrix}_k.$$

- s_j^u — Scale factor for jth MV, in engineering units.
- $w_{i,j}^{\Delta u}$ — Tuning weight for jth MV movement at ith prediction horizon step (dimensionless).

The values n_u, p, s_j^u, and $w_{i,j}^{\Delta u}$ are controller specifications, and are constant. $u(k-1 \mid k) = u(k-1)$, which are the known MVs from the previous control interval. $J_{\Delta u}$ is a function of z_k only.

In addition, a control horizon $m < p$ (or MV blocking) constrains certain MV moves to be zero.

Constraint Violation

In practice, constraint violations might be unavoidable. Soft constraints allow a feasible QP solution under such conditions. MPC controller employs a dimensionless, nonnegative slack variable, ε_k, which quantifies the worst-case constraint violation. (See "Constraints" on page 2-7) The corresponding performance measure is:

$$J_\varepsilon(z_k) = \rho \varepsilon_k^2.$$

Here,

- z_k — QP decision, given by:

$$z_k^T = \begin{bmatrix} u(k \mid k)^T & u(k+1 \mid k)^T & \cdots & u(k+p-1 \mid k)^T \end{bmatrix}_k.$$

- ε_k — Slack variable at control interval k (dimensionless).

- ρ_ϵ — Constraint violation penalty weight (dimensionless).

Alternative Cost Function

You can elect to use the following alternative to the standard cost function:

$$J(z_k) = \sum_{i=0}^{p-1} \left\{ \left[e_y^T(k+i) Q e_y(k+i) \right] + \left[e_u^T(k+i) R_u e_u(k+i) \right] + \left[\Delta u^T(k+i) R_{\Delta u} \Delta u(k+i) \right] \right\} + \rho \quad {}_k^2.$$

Here, Q (n_y-by-n_y), R_u, and $R_{\Delta u}$ (n_u-by-n_u) are positive-semi-definite weight matrices, and:

$$\begin{aligned} e_y(i+k) &= S_y^{-1}\left[r(k+i+1|k) - y(k+i+1|k) \right] \\ e_u(i+k) &= S_u^{-1}\left[u_{target}(k+i|k) - u(k+i|k) \right] \\ \Delta u(k+i) &= S_u^{-1}\left[u(k+i|k) - u(k+i-1|k) \right]. \end{aligned}$$

Also,

- S_y — Diagonal matrix of plant output variable scale factors, in engineering units.
- S_u — Diagonal matrix of MV scale factors in engineering units.
- $r(k+1|k)$ — n_y plant output reference values at the ith prediction horizon step, in engineering units.
- $y(k+1|k)$ — n_y plant outputs at the ith prediction horizon step, in engineering units.
- z_k — QP decision, given by:

$$z_k^T = \left[u(k|k)^T \quad u(k+1|k)^T \quad \cdots \quad u(k+p-1|k)^T \quad {}_k \right].$$

- $u_{target}(k+i|k)$ — n_u MV target values corresponding to $u(k+i|k)$, in engineering units.

Output predictions use the state observer, as in the standard cost function.

The alternative cost function allows off-diagonal weighting, but requires the weights to be identical at each prediction horizon step.

The alternative and standard cost functions are identical if the following conditions hold:

- The standard cost functions employs weights $w_{i,j}^{y}$, $w_{i,j}^{u}$, and $w_{i,j}^{\Delta u}$ that are constant with respect to the index, $i = 1{:}p$.
- The matrices Q, R_u, and $R_{\Delta u}$ are diagonal with the squares of those weights as the diagonal elements.

Constraints

Certain constraints are implicit. For example, a control horizon $m < p$ (or MV blocking) forces some MV increments to be zero, and the state observer used for plant output prediction is a set of implicit equality constraints. Explicit constraints that you can configure are described below.

Bounds on Plant Outputs, MVs, and MV Increments

The most common MPC constraints are bounds, as follows.

$$\frac{y_{j,min}(i)}{s_j^y} - {}_kV_{j,min}^{y}(i) \le \frac{y_j(k+i|k)}{s_j^y} \le \frac{y_{j,max}(i)}{s_j^y} + {}_kV_{j,max}^{y}(i), \quad i = 1:p, \quad j = 1:n_y$$

$$\frac{u_{j,min}(i)}{s_j^u} - {}_kV_{j,min}^{u}(i) \le \frac{u_j(k+i-1|k)}{s_j^u} \le \frac{u_{j,max}(i)}{s_j^u} + {}_kV_{j,max}^{u}(i), \; i = 1:p, \quad j = 1:n_u$$

$$\frac{\Delta u_{j,min}(i)}{s_j^u} - {}_kV_{j,min}^{\Delta u}(i) \le \frac{\Delta u_j(k+i-1|k)}{s_j^u} \le \frac{\Delta u_{j,max}(i)}{s_j^u} + {}_kV_{j,max}^{\Delta u}(i), \; i = 1:p, \quad j = 1:n_u.$$

Here, the V parameters (ECR values) are dimensionless controller constants analogous to the cost function weights but used for constraint softening (see "Constraint Softening"). Also,

- ϵ_k — Scalar QP slack variable (dimensionless) used for constraint softening.
- s_j^y — Scale factor for jth plant output, in engineering units.
- s_j^u — Scale factor for jth MV, in engineering units.
- $y_{j,\min}(i)$, $y_{j,\max}(i)$ — lower and upper bounds for jth plant output at ith prediction horizon step, in engineering units.

- $u_{j,\min}(i)$, $u_{j,\max}(i)$ — lower and upper bounds for jth MV at ith prediction horizon step, in engineering units.
- $\Delta u_{j,\min}(i)$, $\Delta u_{j,\max}(i)$ — lower and upper bounds for jth MV increment at ith prediction horizon step, in engineering units.

Except for the slack variable non-negativity condition, all of the above constraints are optional and are inactive by default (i.e., initialized with infinite limiting values). To include a bound constraint, you must specify a finite limit when you design the controller.

QP Matrices

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

Prediction

Assume that the disturbance models described in "Input Disturbance Model" is unit gain, for example, $d(k)=n_d(k)$ is a white Gaussian noise). You can denote this problem as

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

Then, the prediction model is:

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

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

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

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

This equation gives the solution

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

where

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

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

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

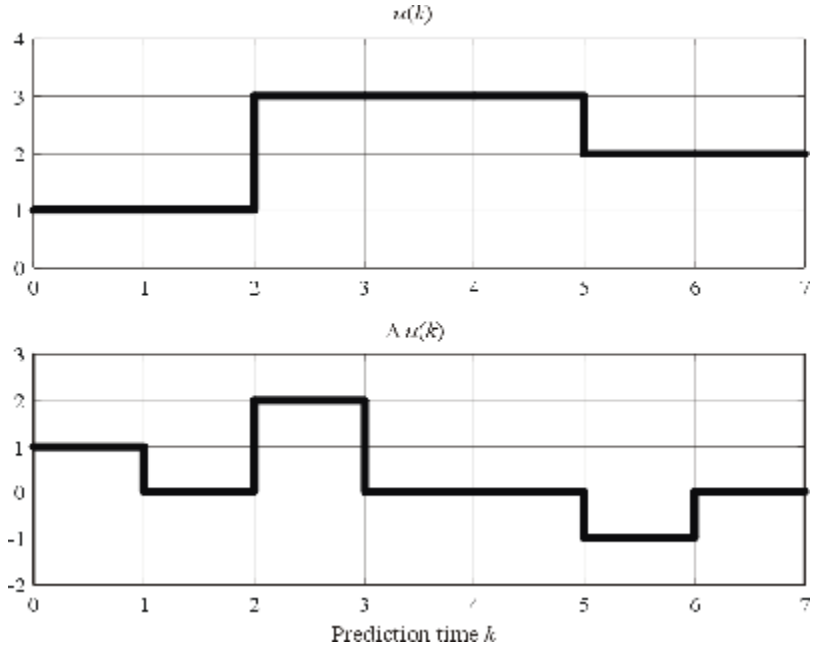
Optimization Variables

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

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

where J_M depends on the choice of blocking moves. Together with the slack variable ε, vectors $z_0, \ldots, z_{m-1}$ constitute the free optimization variables of the optimization problem. In the case of systems with a single manipulated variables, $z_0, \ldots, z_{m-1}$ are scalars.

Consider the blocking moves depicted in the following graph.



Blocking Moves: Inputs and Input Increments for moves = [2 3 2]

This graph corresponds to the choice `moves=[2 3 2]`, or, equivalently, $u(0)=u(1)$, $u(2)=u(3)=u(4)$, $u(5)=u(6)$, $\Delta u(0)=z0$, $\Delta u(2)=z1$, $\Delta u(5)=z2$, $\Delta u(1)=\Delta u(3)=\Delta u(4)=\Delta u(6)=0$.

Then, the corresponding matrix J_M is

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

Cost Function

- "Standard Form" on page 2-11
- "Alternative Cost Function" on page 2-12

Standard Form

The function to be optimized is

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

where

$$W_u=\mathrm{diag}\left(w_{0,1}^u,w_{0,2}^u,\ldots,w_{0,n_u}^u,\ldots,w_{p-1,1}^u,w_{p-1,2}^u,\ldots,w_{p-1,n_u}^u\right)$$
$$W_{\Delta u}=\mathrm{diag}\left(w_{0,1}^{\Delta u},w_{0,2}^{\Delta u},\ldots,w_{0,n_u}^{\Delta u},\ldots,w_{p-1,1}^{\Delta u},w_{p-1,2}^{\Delta u},\ldots,w_{p-1,n_u}^{\Delta u}\right)$$
$$W_y=\mathrm{diag}\left(w_{1,1}^y,w_{1,2}^y,\ldots,w_{1,n_y}^y,\ldots,w_{p,1}^y,w_{p,2}^y,\ldots,w_{p,n_y}^y\right)$$

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

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

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

Alternative Cost Function

If you are using the alternative cost function shown in "Alternative Cost Function" on page 2-6, Equation 2-3, then Equation 2-2 is replaced by the following:

$$W_u = \text{blkdiag}(R_u, \ldots, R_u)$$
$$W_{\Delta u} = \text{blkdiag}(R_{\Delta u}, \ldots, R_{\Delta u})$$
$$W_y = \text{blkdiag}(Q, \ldots, Q)$$

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

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

Constraints

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

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

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

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

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

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

Unconstrained Model Predictive Control

The optimal solution is computed analytically

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

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

More About

- "Adjusting Disturbance and Noise Models" on page 2-14
- "Time-Varying Weights and Constraints" on page 2-20
- "Terminal Weights and Constraints" on page 2-22

Adjusting Disturbance and Noise Models

Model Predictive Control requires the following in order to reject unknown disturbances effectively:

- Disturbance modeling tailored to the application
- Feedback from the measurements to update controller state estimates

This section topic guidance for these issues, focusing on design-tool options.

In this section...

Overview

MPC attempts to predict how known and unknown events will affect the plant output variables (OVs). Known events are changes in the measured plant input variables (MV and MD inputs). The controller's plant model predicts their impact (see "MPC Modeling"), and such predictions can be quite accurate.

The impacts of unknown events appear as errors in the predictions of known events. These errors are, by definition, impossible to predict accurately. An ability to anticipate trends can improve disturbance rejection, however.

For example, suppose the control system has been operating at a near-steady condition with all measured OVs near their predicted values. There are no known events, but one or more of these OVs suddenly deviates from its prediction. How should the controller react? Its disturbance and measurement models allow you to provide guidance.

Output Disturbance Model

For the moment, suppose your plant model includes no unmeasured disturbance inputs. The MPC controller then models unknown events using an *output disturbance model*.

As shown in “MPC Modeling”, the output disturbance model is independent of the plant model, and its output adds directly to that of the plant model.

The design tool requires one of the following assumptions regarding the disturbances affecting a given plant OV (signal-by-signal option):

- Any prediction error is a realization of white noise with zero mean. This assumption implies that the of impact of noise is short-lived, calling for a modest, short-term controller response.
- Any prediction error is due to a step-like disturbance (the default), which lasts indefinitely, maintaining a roughly constant magnitude. This assumption calls for a more aggressive, sustained controller response.
- Any prediction error is due to a ramp-like disturbance, which lasts indefinitely and tending to grow with time. This assumption calls for an even more aggressive controller response.

Each assumption can be represented by a model in which white noise with unit variance, zero mean enters a SISO dynamic system consisting of one of the following:

- A static gain, for white noise disturbance
- An integrator in series with a static gain, for step-like disturbance
- Two integrators in series with a static gain, for ramp-like disturbance

The design tool also allows you to specify the white noise input magnitude, overriding the assumption of unit variance. As you increase this, the controller will respond more aggressively to a given prediction error. (Your magnitude specification determines the gain value associated with your chosen model form: white, step, or ramp.)

Measurement Noise Model

MPC also attempts to distinguish disturbances, which require a controller response, from measurement noise, which the controller should ignore. To guide it, you specify the expected measurement noise magnitude and character. The design tool options parallel the output disturbance model case, but the character must be either white (random) or step (sustained). In nearly all applications, the white noise option should provide adequate performance and is the default.

When you include a measurement noise model, the controller considers each prediction error to be a combination of disturbance and noise effects. Qualitatively, as you increase

the specified noise magnitude, the controller attributes a larger fraction of each prediction error to noise, and it responds less aggressively. Ultimately, the controller stops responding to prediction errors and only changes its MVs when you change the OV or MV reference signals.

Input Disturbance Model

When your plant model includes unmeasured disturbance (UD) inputs, the controller can employ an *input disturbance model* in addition to the standard output disturbance model. The former provides more flexibility and is generated automatically by default. (If the chosen input disturbance model does not appear to allow complete elimination of sustained disturbances, the Toolbox adds an output disturbance model by default.)

As shown in "MPC Modeling", the input disturbance model consists of one or more white noise signals (unit variance, zero mean) entering a dynamic system, whose outputs are the plant model's UD inputs.

As with the output disturbance model, the design tool allows you to specify that a UD signal generated by the input disturbance model has a white (random), step (sustained – the default), or ramp (growing) character. In contrast to the output disturbance model, these UD disturbances then affect the outputs in a more complex way as they pass through the plant model dynamics.

A popular approach is to model unknown events as disturbances adding to the plant MVs. (These are termed *load disturbances* in many texts, and are realistic in that some unknown events are failures to set the MVs to the values requested by the controller.) You can create a load disturbance model as follows:

1. Begin with an LTI plant model in which all inputs are known (MVs and MDs). Suppose this model is called `Plant`.
2. Obtain the state-space matrices of `Plant`. For example:

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

3. Suppose there are n_u MVs. Set B_u = columns of B corresponding to the MVs. Also set D_u = columns of D corresponding to the MVs.
4. Redefine the plant model to include nu additional inputs. For example:

   ```
   set(Plant,'b',[B, Bu],'d',[D, Du])
   ```

5. Use `setmpcsignals`, or set the `Plant.InputGroup` property, to indicate that the new inputs are unmeasured disturbances.

This procedure adds load disturbance inputs without increasing the number of states in the plant model.

By default, given a plant model containing load disturbances, the Model Predictive Control Toolbox software creates an input disturbance model that generates n_{ym} step-like load disturbances. If $n_{ym} > n_u$, it will also create an output disturbance model with integrated white noise adding to $(n_{ym} - n_u)$ measured outputs. If $n_{ym} < n_u$, the last $(n_u - n_{ym})$ load disturbances will be zero by default. You can modify these defaults using the design tool.

Restrictions

As discussed in "Controller State Estimation" on page 2-29, the plant, disturbance, and noise models combine to form a state observer, which must be detectable using the measured plant outputs. If not, the software displays a command-window error message when you attempt to use the controller.

This restricts the form of the disturbance and noise models. If these models are other than a static gain, their model states must be detectable.

For example, an integrated white noise disturbance adding to an unmeasured OV would be undetectable. The design tool prevents you from choosing such a model. Similarly, the number of measured disturbances, n_{ym}, limits the number of step-like UD inputs from an input disturbance model.

By default, the Model Predictive Control Toolbox software creates detectable models. If you modify the default assumptions (or change n_{ym}) and encounter a detectability error, you can revert to the default case.

Disturbance Rejection Tuning

The following suggestions derive from the above qualitative description of the disturbance and measurement noise models.

- Prior to any controller tuning, define scale factors for each plant input and output variable (see "Specifying Scale Factors"). In the context of disturbance and noise modeling, this makes the default assumption of unit-variance white noise inputs more likely to yield good performance.
- Allow the toolbox to create default disturbance and measurement models.

- After tuning the cost function weights (see "Tuning Weights"), test your controller's response to an unmeasured disturbance input other than a step disturbance at the plant output. Specifically, if your plant model includes UD inputs, simulate a disturbance using one or more of these. Otherwise, simulate one or more load disturbances, i.e., a step disturbance added to a designated MV. Both the design tool and the `sim` command support such simulations.
- If the response in the simulations seems too sluggish, try one or more of the following:
  - Increase all disturbance model gains by a multiplicative factor (e.g., 5). In the design tool, do this by increasing the specified magnitude of each disturbance. If this helps but is insufficient, repeat.
  - Decrease the measurement noise gains by a multiplicative factor. In the design tool, decrease the specified measurement noise magnitude. If this helps but is insufficient, repeat.
  - Change the character of one or more disturbances (from white to step, or from step to ramp). Note, however, that if a disturbance is white by default, changing it to step might violate the state observer detectability restriction.
- If the response is too aggressive, and in particular, if the controller is not robust when its prediction of known events is inaccurate, try reversing the above steps.

Related Examples

- "Design Controller Using the Design Tool"

More About

- "MPC Modeling"
- "Controller State Estimation" on page 2-29

Custom State Estimation

The Model Predictive Control Toolbox software allows the following alternatives to the default state estimation approach:

- You can override the default Kalman gains, L and M. Obtain the default values using `getEstimator`. Then, use `setEstimator` to override those values. These commands assume that the columns of L and M are in the engineering units for the measured plant outputs. Internally, the software converts them to dimensionless form.
- You can use the custom estimation option. This skips all Kalman gain calculations. When the controller operates, at each control interval you must use an external procedure to estimate the controller states, `xc(k|k)`, providing this to the controller.

Related Examples

- Using Custom State Estimation

More About

- “Controller State Estimation” on page 2-29

Time-Varying Weights and Constraints

In this section...

Time-Varying Weights

As explained in "Optimization Problem" on page 2-2, the w^y, w^u and $w^{\Delta u}$ weights can change from one step in the prediction horizon to the next. Such a *time-varying weight* is an array containing p rows, where p is the prediction horizon, and either n_y or n_u columns (number of OVs or MVs).

Using time-varying weights provides additional tuning possibilities. However, it complicates tuning. Recommended practice is to use constant weights unless your application includes unusual characteristics. For example, an application requiring terminal weights must employ time-varying weights. See "Terminal Weights and Constraints" on page 2-22.

Time-Varying Constraints

When bounding an MV, OV, or MV increment, you have the option to use a different bound value at each prediction-horizon step. To do so, specify the bound as a vector of up to p values, where p is the prediction horizon length (number of control intervals). If you specify $n < p$ values, the nth value applies for the remaining $p - n$ steps.

You can remove constraints at selected steps by specifying `Inf` (or `-Inf`).

If plant delays prevent the MVs from affecting an OV during the first d steps of the prediction horizon and you must include bounds on that OV, leave the OV unconstrained for the first d steps.

Related Examples

- Varying Input and Output Constraints

More About

- "Optimization Problem" on page 2-2

Terminal Weights and Constraints

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

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

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

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

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

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

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

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

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

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

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

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

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

1. Augment the model (Equation 2-7) to include the weighted terminal states as auxiliary outputs:

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

   where Q_c is the Cholesky factorization of Q_p such that $Q_p = Q_c^T Q_c$.
2. Define the auxiliary outputs y_{aug} as unmeasured, and specify zero weight to them.
3. Specify unity weight on y_{aug} at the last step in the prediction horizon using `setterminal`.

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

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

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

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

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

Related Examples

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

Constraints on Linear Combinations of Inputs and Outputs

You can also constrain linear combinations of plant input and output variables. For example, you might want a particular manipulated variable (MV) to be greater than a linear combination of two other MVs. The general form of such constraints is the following:

$$Eu(k+i|k) + Fy(k+i|k) + Sv(k+i\,|\,k) \le G + {}_{k}V.$$

Here,

- $\in_k$ — QP slack variable used for constraint softening (See "Constraint Softening").
- $u(k+i\,|\,k)$ — n_u MV values, in engineering units
- $y(k+i\,|\,k)$ — n_y predicted plant outputs, in engineering units
- $v(k+i\,|\,k)$ — n_v measured plant disturbance inputs, in engineering units
- E, F, S, G, and V are constants

As with the QP cost function, output prediction using the state observer makes these constraints a function of the QP decision.

Custom constraints are dimensional by default. It is good practice to define the constant coefficients (E, F, S, G, V) such that each term is dimensionless and of order unity. This requires application-specific analysis.

See Also

`getconstraint` | `setconstraint`

Related Examples

- Using Custom Input and Output Constraints
- Nonlinear Blending Process with Custom Constraints

More About

- "Optimization Problem" on page 2-2

Manipulated Variable Blocking

Blocking is an alternative to the simpler control horizon concept (see “Choosing Sample Time and Horizons”). It has many of the same benefits. It also provides more tuning flexibility and potential to smooth MV adjustments. A recommended approach to blocking is as follows:

- Divide the prediction horizon into 3-5 blocks.
- Try the following alternatives
  - Equal block sizes (one-fifth to one-third of the prediction horizon, p)
  - Block sizes increasing. Example, with $p = 20$, three blocks of duration 3, 7 and 10 intervals.

Test the resulting controller in the same way as you test cost function weights. See “Tuning Weights”.

Related Examples

- “Design Controller for Plant with Delays”

QP Solver

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

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

such that

$$Ax \le b$$

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

The toolbox uses the KWIK algorithm [1] to solve the QP problem, which requires the Hessian to be positive definite. In the very first control step, KWIK uses a *cold start*, in which the initial guess is the unconstrained solution described in "Unconstrained Model Predictive Control" on page 2-13. If this x satisfies the constraints, it is the optimal QP solution, x^*, and the algorithm terminates. Otherwise this means that at least one of the linear inequality constraints must be satisfied as an equality. In this case, KWIK uses an efficient, numerically robust strategy to determine the active constraint set satisfying the standard optimality conditions. In the following control steps, KWIK uses a *warm start*. In this case, the active constraint set determined at the previous control step becomes the initial guess for the next.

Although KWIK is robust, you should consider the following:

- One or more linear constraints might be violated slightly due to numerical round-off errors. The toolbox employs a nonadjustable relative tolerance. This tolerance allows a constraint to be violated by 10^{-6} times the magnitude of each term. Such violations are considered normal and do not generate warning messages.
- The toolbox also uses a nonadjustable tolerance when it tests a solution for optimality.
- The search for the active constraint set is an iterative process. If the iterations reach a problem-dependent maximum, the algorithm terminates.

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

In the last two situations, with an abnormal outcome to the search, the controller will retain the last successful control output. For more information, see, the `mpcmove` command. You can detect an abnormal outcome and override the default behavior as you see fit.

References

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

More About

- "Optimization Problem" on page 2-2

Controller State Estimation

In this section...

Controller State Variables

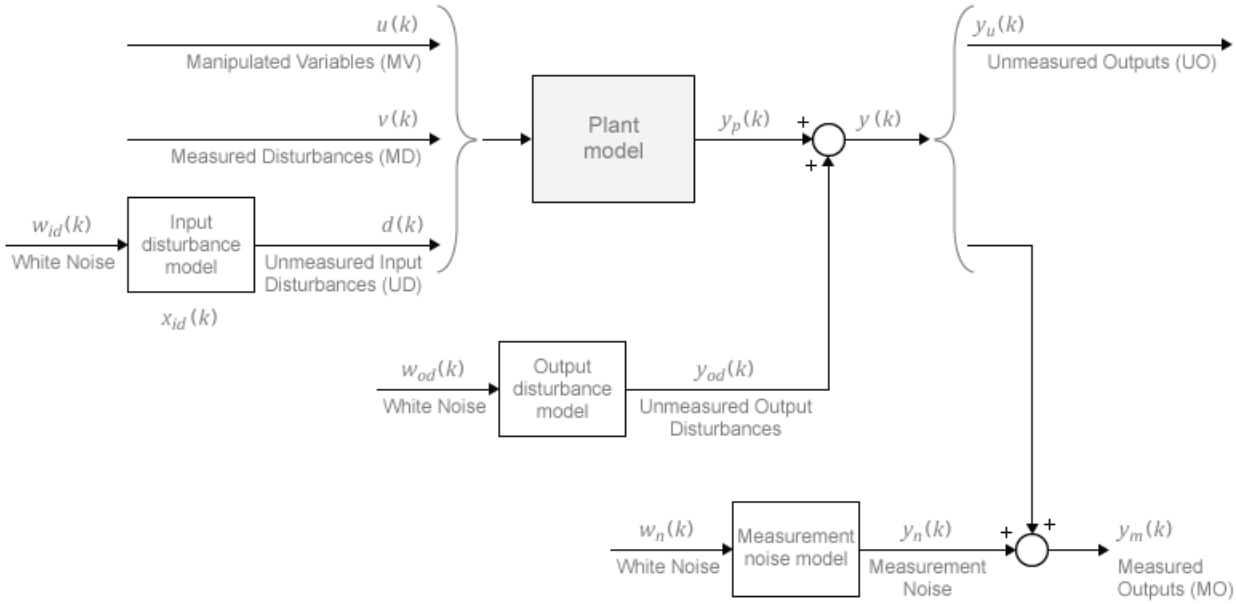
As the controller operates, it uses its current state, x_c, as the basis for predictions. By definition, the state vector is the following:

$$x_c^T(k) = \begin{bmatrix} x_p^T(k) & x_{id}^T(k) & x_{od}^T(k) & x_n^T(k) \end{bmatrix}.$$

Here,

- x_c is the controller state, comprising $n_{xp} + n_{xid} + n_{xod} + n_{xn}$ state variables.
- x_p is the plant model state vector, of length n_{xp}.
- x_{id} is the input disturbance model state vector, of length n_{xid}.
- x_{od} is the output disturbance model state vector, of length n_{xod}.
- x_n is the measurement noise model state vector, of length n_{xn}.

Thus, the variables comprising x_c represent the models appearing in the following diagram of the MPC system.



Some of the state vectors may be empty. If not, they appear in the sequence defined within each model.

By default, the controller updates its state automatically using the latest plant measurements. See "State Estimation" on page 2-31 for details. Alternatively, the custom state estimation feature allows you to update the controller state using an external procedure, and then supply these values to the controller. See "Custom State Estimation" on page 2-19 for details.

State Observer

Combination of the models shown in the diagram yields the state observer:

$$x_c(k+1) = Ax_c(k) + Bu_o(k)$$
$$y(k) = Cx_c(k) + Du_o(k).$$

MPC controller uses the state observer in the following ways:

- To estimate values of unmeasured states needed as the basis for predictions (see "State Estimation" on page 2-31).

- To predict how the controller's proposed manipulated variable (MV) adjustments will affect future plant output values (see "Output Variable Prediction" on page 2-34).

The observer's input signals are the dimensionless plant manipulated and measured disturbance inputs, and the white noise inputs to the disturbance and noise models:

$$u_o^T(k) = \begin{bmatrix} u^T(k) & v^T(k) & w_{id}^T(k) & w_{od}^T(k) & w_n^T(k) \end{bmatrix}.$$

The observer's outputs are the n_y dimensionless plant outputs.

In terms of the parameters defining the four models shown in the diagram, the observer's parameters are:

$$A = \begin{bmatrix} A_p & B_{pd}C_{id} & 0 & 0 \\ 0 & A_{id} & 0 & 0 \\ 0 & 0 & A_{od} & 0 \\ 0 & 0 & 0 & A_n \end{bmatrix}, \quad B = \begin{bmatrix} B_{pu} & B_{pv} & B_{pd}D_{id} & 0 & 0 \\ 0 & 0 & B_{id} & 0 & 0 \\ 0 & 0 & 0 & B_{od} & 0 \\ 0 & 0 & 0 & 0 & B_n \end{bmatrix},$$

$$C = \begin{bmatrix} C_p & D_{pd}C_{id} & C_{od} & \begin{bmatrix} C_n \\ 0 \end{bmatrix} \end{bmatrix}, \quad D = \begin{bmatrix} 0 & D_{pv} & D_{pd}D_{id} & D_{od} & \begin{bmatrix} D_n \\ 0 \end{bmatrix} \end{bmatrix}.$$

Here, the plant and output disturbance models are resequenced so that the measured outputs precede the unmeasured outputs.

State Estimation

In general, the controller states are unmeasured and must be estimated. By default, the controller uses a steady state Kalman filter that derives from the state observer. (See "State Observer" on page 2-30.)

At the beginning of the kth control interval, the controller state is estimated with the following steps:

1 Obtain the following data:

- $x_c(k \mid k-1)$ — Controller state estimate from previous control interval, $k-1$
- $u^{act}(k-1)$ — Manipulated variable (MV) actually used in the plant from $k-1$ to k (assumed constant)

- $u^{opt}(k–1)$ — Optimal MV recommended by MPC and assumed to be used in the plant from $k–1$ to k
- $v(k)$ — Current measured disturbances
- $y_m(k)$ — Current measured plant outputs
- B_u, B_v — Columns of observer parameter B corresponding to $u(k)$ and $v(k)$ inputs
- C_m — Rows of observer parameter C corresponding to measured plant outputs
- D_{mv} — Rows and columns of observer parameter D corresponding to measured plant outputs and measured disturbance inputs
- L, M — Constant Kalman gain matrices

Plant input and output signals are scaled to be dimensionless prior to use in calculations.

2 Revise $x_c(k\,|\,k–1)$ when $u^{act}(k–1)$ and $u^{opt}(k–1)$ are different:

$$x_c^{rev}(k|k-1) = x_c(k|k-1) + B_u\left[u^{act}(k-1) - u^{opt}(k-1)\right].$$

3 Compute the innovation:

$$e(k) = y_m(k) - \left[C_m x_c^{rev}(k|k-1) + D_{mv}v(k)\right].$$

4 Update the controller state estimate to account for the latest measurements.

$$x_c(k|k) = x_c^{rev}(k|k-1) + Me(k).$$

Then, the software uses the current state estimate $x_c(k\,|\,k)$ to solve the quadratic program at interval k. The solution is $u^{opt}(k)$, the MPC-recommended manipulated-variable value to be used between control intervals k and k+1.

Finally, the software prepares for the next control interval assuming that the unknown inputs, $w_{id}(k)$, $w_{od}(k)$, and $w_n(k)$ assume their mean value (zero) between times k and k+1. The software predicts the impact of the known inputs and the innovation as follows:

$$x_c(k+1\,|\,k) = Ax_c(k\,|\,k) + B_u u^{opt}(k) + B_v v(k) + Le(k).$$

Built-in Steady-State Kalman Gains Calculation

Model Predictive Control Toolbox software uses the `kalman` command to calculate Kalman estimator gains L and M. The following assumptions apply:

- State observer parameters A, B, C, D are time-invariant.
- Controller states, x_c, are detectable. (If not, or if the observer is numerically close to undetectability, the Kalman gain calculation fails, generating an error message.)
- Stochastic inputs $w_{id}(k)$, $w_{od}(k)$, and $w_n(k)$ are independent white noise, each with zero mean and identity covariance.
- Additional white noise $w_u(k)$ and $w_v(k)$ with the same characteristics adds to the dimensionless $u(k)$ and $v(k)$ inputs respectively. This improves estimator performance in certain cases, such as when the plant model is open-loop unstable.

Without loss of generality, set the $u(k)$ and $v(k)$ inputs to zero. The effect of the stochastic inputs on the controller states and measured plant outputs is:

$$\begin{aligned} x_c(k+1) &= Ax_c(k) + Bw(k) \\ y_m(k) &= C_m x_c(k) + D_m w(k). \end{aligned}$$

Here,

$$w^T(k) = \begin{bmatrix} w_u^T(k) & w_v^T(k) & w_{id}^T(k) & w_{od}^T(k) & w_n^T(k) \end{bmatrix}.$$

Inputs to the `kalman` command are the state observer parameters A, C_m, and the following covariance matrices:

$$\begin{aligned} Q &= E\left\{Bww^TB^T\right\} = BB^T \\ R &= E\left\{D_m ww^T D_m^T\right\} = D_m D_m^T \\ N &= E\left\{Bww^T D_m^T\right\} = BD_m^T. \end{aligned}$$

Here, $E\{...\}$ denotes the expectation.

Output Variable Prediction

Model Predictive Control requires prediction of noise-free future plant outputs used in optimization. This is a key application of the state observer (see "State Observer" on page 2-30).

In control interval k, the required data are as follows:

- p — Prediction horizon (number of control intervals, which is greater than or equal to 1)
- $x_c(k|k)$ — Controller state estimates (see "State Estimation" on page 2-31)
- $v(k)$ — Current measured disturbance inputs (MDs)
- $v(k+i|k)$ — Projected future MDs, where i=1:p–1. If you are not using MD previewing, then $v(k+i|k) = v(k)$.
- A, B_u, B_v, C, D_v — State observer constants, where B_u, B_v, and D_v denote columns of the B and D matrices corresponding to inputs u and v. D_u is a zero matrix because of no direct feedthrough

Predictions assume that unknown white noise inputs are zero (their expectation). Also, the predicted plant outputs are to be noise-free. Thus, all terms involving the measurement noise states disappear from the state observer equations. This is equivalent to zeroing the last nxn elements of $x_c(k|k)$.

Given the above data and simplifications, for the first step the state observer predicts:

$$x_c(k+1|k) = Ax_c(k|k) + B_u u(k|k) + B_v v(k).$$

Continuing for successive steps, i = 2:p, the state observer predicts:

$$x_c(k+i|k) = Ax_c(k+i-1|k) + B_u u(k+i-1|k) + B_v v(k+i-1|k).$$

At any step, i = 1:p, the predicted noise-free plant outputs are:

$$y(k+i|k) = Cx_c(k+i|k) + D_v v(k+i|k).$$

All of these equations employ dimensionless plant input and output variables. See "Specifying Scale Factors". The equations also assume zero offsets. Inclusion of nonzero offsets is straightforward.

For faster computations, the MPC controller uses an alternative form of the above equations in which constant terms are computed and stored during controller initialization. See “QP Matrices” on page 2-8.

More About

- “MPC Modeling”
- “Optimization Problem” on page 2-2

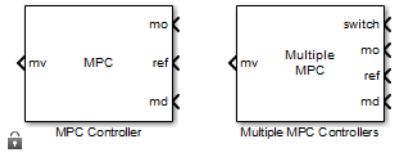
3

Model Predictive Control Simulink Library

MPC Library

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

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



MPC Simulink Library

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

MPC Controller Block

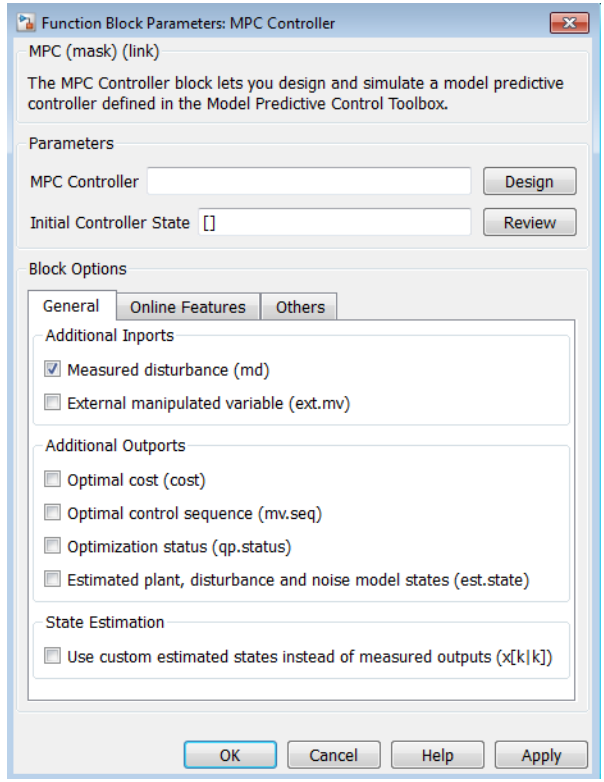
In this section...

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

MPC Controller Block Mask

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

- Specifying required parameters.
- Enabling optional inports and outputs.
- Connecting appropriate signals.



MPC Controller Block Mask

MPC Controller Parameters

Specify the MPC Controller

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

- Type the name of an MPC object saved in your workspace. To review its settings before running a simulation, click Design to load the named object into the MPC design tool.
- If your installation includes Simulink Control Design™ software, connect the MPC Controller block to the plant it controls. Leaving the MPC controller field empty, click Design. The block constructs a default MPC object by linearizing the plant at the

default operating point and exports this defaults controller to the base workspace. If the default operating point is not the one you want, see "Importing a Plant Model" for help with importing a new plant model.

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

Indicate Initial Controller State

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

Connect Signals

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

Required inputs

Connect the following to the indicated controller inports:

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

You can also preview reference signals at run-time. To activate reference previewing, supply the reference input as an Nxn_y signal, where $1 < N \leq p$, and p is the prediction horizon length. The rows 1 to N define the reference values for time instants t_k+1 to t_k+N

in the prediction horizon. If N< p, the last row is used for steps t_k+n+1 to t_k+p. See the mpcpreview example.

Required outport

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

Sample Time

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

Optional Ports

Feedforward compensation for measured disturbances

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

- $n_{md} \geq 1$ is the number of measured disturbances coming from the plant
- $N \geq 1$ is the number of sampling instants for which you are supplying measured disturbance values.

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

Externally supplied MV signals

Ideally, the manipulated variable values used in the plant are always identical to those specified by the controller's mv outport. The controller makes this assumption by default. In practice, however, unexpected constraints, disturbances, and plant nonlinearities can prevent these values from being exactly alike. If the actual values of manipulated

variables are measured and fed back, the controller's predictions improve. This feature can also smooth the transition between manual and automatic operation. For more information, see "Bumpless Transfer Between Manual and Automatic Operations".

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

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

Input and output limits

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

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

- `umin` defines n_{mv} lower bounds on the manipulated variables (the controller's `mv` outport).
- `umax` is a corresponding vector of n_{mv} upper bounds.
- `ymin` defines n_y lower bounds on the output variables (the same variables to which the ref inport applies).
- `ymax` is a corresponding vector of n_y upper bounds.

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

Disabling optimization

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

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

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

For more information and examples, see “Bumpless Transfer Between Manual and Automatic Operations”.

Monitoring the optimal cost

This option allows you to monitor the objective function value (optimal cost) obtained by solving the controller's quadratic program (QP). See “Optimization Problem” for more information.

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

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

Monitoring the optimal control sequence

To monitor the controller's planned sequence of future adjustments, select Optimal control sequence, and click **Apply**. This option adds an outport labeled `mv.seq`. At each control instant, the outport contains a `p X` n_{mv} matrix signal. The rows correspond to the `p` steps of the prediction horizon, and the columns to the n_{mv} manipulated variables. The first row represents current time, `k=0`. The last row represents time `k+p-1`.

Monitoring the QP status

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

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

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

Possible abnormal terminations are:

- **= 0**: The QP could not be solved in the maximum number of iterations specified in the controller definition.
- **= -1**: The QP solver detected an infeasible QP, that is, it was impossible to satisfy all the hard constraints imposed in the controller definition.

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

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

Online tuning

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

Weights on plant outputs

The added inport is labeled `y.wt`. Connect a vector signal, length n_y, which defines the nonnegative real weight on reference tracking for each of the n_y output variables. This signal overrides the controller's `MPCobj.Weights.OV` property. A larger weight increases the importance of accurate tracking for the corresponding output variable.

Weights on manipulated variables rate

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

Weight on overall constraint softening

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

Input Signals

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

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

Enable externally supplied MV signals allows you to keep the controller informed of the *actual* manipulated variable values. Ideally, the actual manipulated variables are those specified by the controller block output `mv`. In practice, unexpected constraints, disturbances, or plant nonlinearities can modify the values actually implemented in the plant. If the actual values are known and fed back to the controller, its predictions improve. This feature can also improve the transition between manual and automatic operation. See "Bumpless Transfer Between Manual and Automatic Operations" on page 4-40.

Enable input and output limits allows you to specify constraints that vary during a simulation. Otherwise, the block uses the constant constraint values stored within its MPC Controller object. The example `mpcvarbounds` shows how this option works. It enables inports for lower and upper bounds on the manipulated variables (inports `umin` and `umax`) and lower and upper bounds on the controlled outputs (inports `ymin` and `ymax`). An unconnected constraint inport causes the corresponding variable to be unconstrained.

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

If you select the switching option, the **Enable externally supplied MV signals** option must also be activated. See "Bumpless Transfer Between Manual and Automatic Operations" on page 4-40 for an example application.

Output Signals

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

There are two optional outputs:

- The **Enable optimal cost outport** option adds an outport labeled cost, which contains the value of the optimal objective function obtained when calculating the

manipulated variables. See "Optimization Problem" on page 2-2 for the various forms this can take. If the optimization problem is infeasible, i.e., some constraints can't be satisfied or the solver experiences numerical difficulties, the returned cost is –1 .

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

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

Look Ahead and Signals from the Workspace

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

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

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

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

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

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

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

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

Initialization

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

Generate Code and Deploy Controller to Real-Time Targets

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

- Simulink Coder™
- Embedded Coder®
- Simulink PLC Coder™
- Simulink Real-Time™

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

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

See Also

MPC Controller | Multiple MPC Controllers | `review`

Related Examples

- "Simulation and Code Generation Using Simulink Coder"
- "Simulation and Structured Text Generation Using PLC Coder"

Multiple MPC Controllers Block

In this section...

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

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

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

Limitations

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

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

Examples

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

Relationship of Multiple MPC Controllers to MPC Controller Block

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

Listing the controllers

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

Designing the controllers

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

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

Defining controller switching

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

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

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

Improving prediction accuracy

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

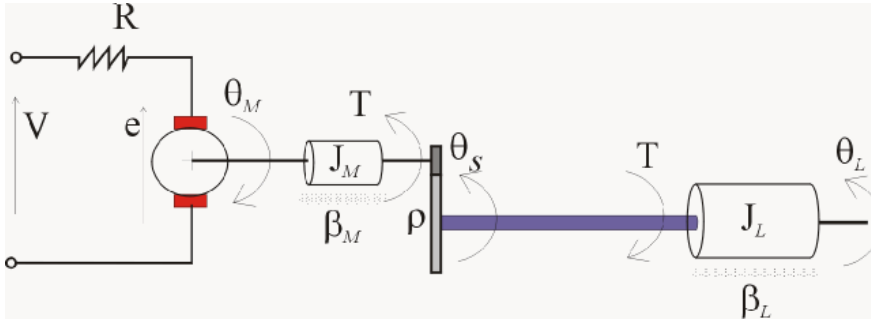
Case-Study Examples

Servomechanism Controller

In this section...

System Model

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



Position Servomechanism Schematic

The differential equations representing this system are

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

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

where V is the applied voltage, T is the torque acting on the load, $\omega_L = \dot{\theta}_L$ is the load's angular velocity, $\omega_M = \dot{\theta}_M$ is the motor shaft's angular velocity, and the other symbols

represent constant parameters (see Parameters Used in the Servomechanism Model for more information on these).

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

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

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

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

Parameters Used in the Servomechanism Model

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

Control Objectives and Constraints

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

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

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

Also, the applied voltage must stay within the range

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

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

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

Defining the Plant Model

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

```
% DC-motor with elastic shaft
%
%Parameters (MKS)
%-------------------------------------------------------------
Lshaft=1.0;          %Shaft length
dshaft=0.02;         %Shaft diameter
shaftrho=7850;       %Shaft specific weight (Carbon steel)
G=8150O*1e6;         %Modulus of rigidity
tauam=50*1e6;        %Shear strength
Mmotor=100;          %Rotor mass
Rmotor=.1;           %Rotor radius
Jmotor=.5*Mmotor*Rmotor^2;       %Rotor axial moment of inertia
Bmotor=0.1;          %Rotor viscous friction coefficient (A CASO)
R=20;                %Resistance of armature
Kt=10;               %Motor constant
gear=20;             %Gear ratio
Jload=50*Jmotor;     %Load inertia
Bload=25;            %Load viscous friction coefficient
Ip=pi/32*dshaft^4;                    %Polar momentum of shaft (circular) section
Kth=G*Ip/Lshaft;                     %Torsional rigidity (Torque/angle)
Vshaft=pi*(dshaft^2)/4*Lshaft;         %Shaft volume
Mshaft=shaftrho*Vshaft;                %Shaft mass
Jshaft=Mshaft*.5*(dshaft^2/4);         %Shaft moment of inertia
JM=Jmotor;
JL=Jload+Jshaft;
Vmax=tauam*pi*dshaft^3/16;       %Maximum admissible torque
Vmin=-Vmax;

%Input/State/Output continuous time form
%-------------------------------------------------------------
```

```
AA=[0             1             0                 0;
    -Kth/JL       -Bload/JL     Kth/(gear*JL)     0;
    0             0             0                 1;
    Kth/(JM*gear) 0             -Kth/(JM*gear^2)  -(Bmotor+Kt^2/R)/JM];

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

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

Controller Design Using MPCTOOL

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

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

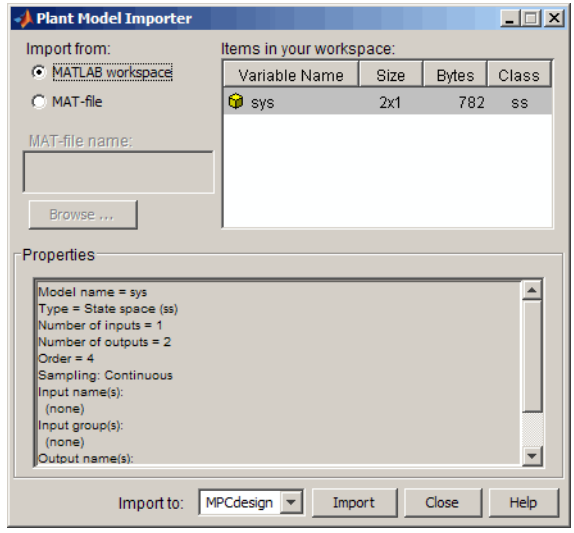
Opening MPCTOOL and Importing a Model

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

```
mpctool
```

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

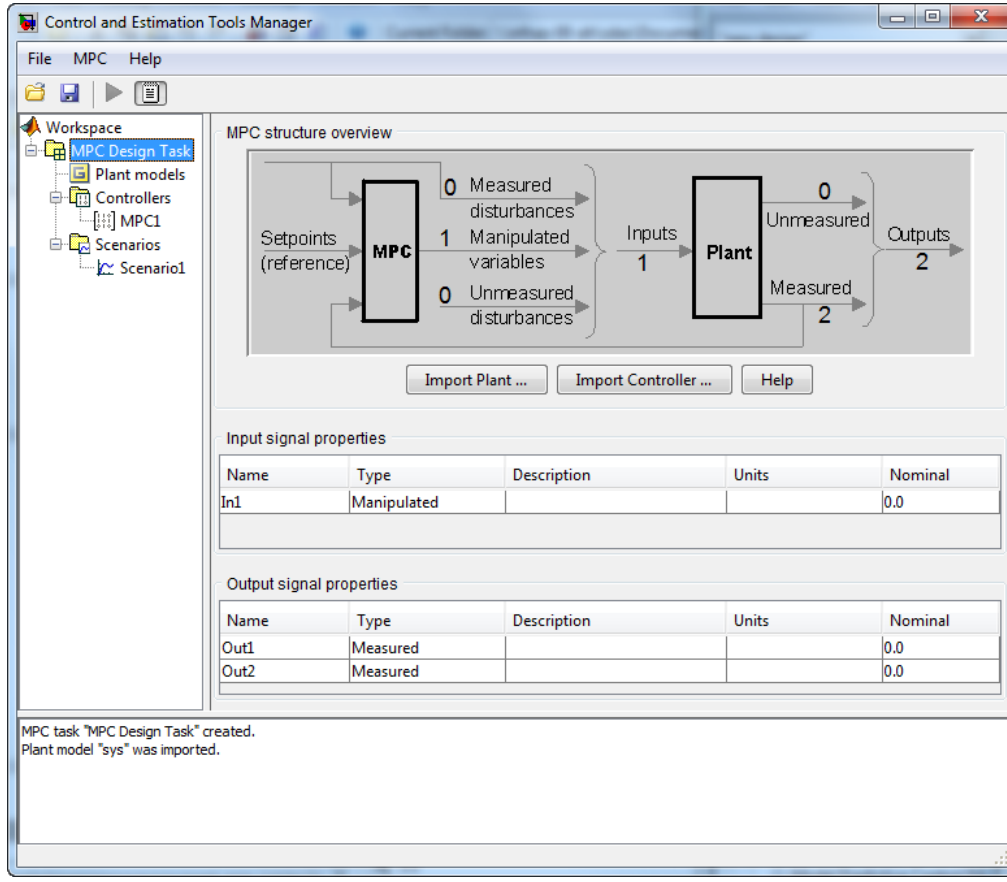
By default, the **Import from** option buttons are set to import from the MATLAB workspace, and the box at the upper right lists all LTI models defined there. In the following figure, `sys` is the only available model, and it is selected. The **Properties** area lists the selected model's key attributes.



Import Dialog Box with the Servomechanism Model Selected

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

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



Design Tool After Importing the Plant Model

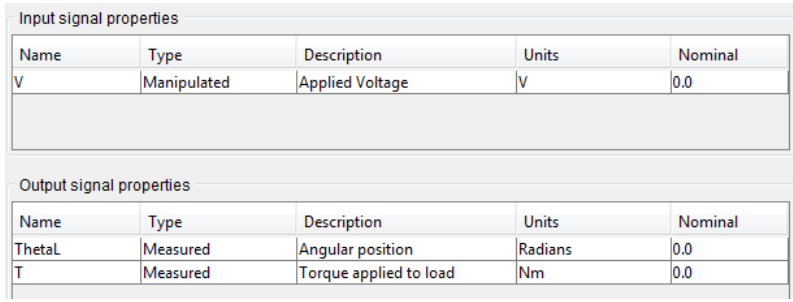
Specifying Signal Properties

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

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

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

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



Design Tool After Specifying Signal Properties

Navigation Using the Tree View

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

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). 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.

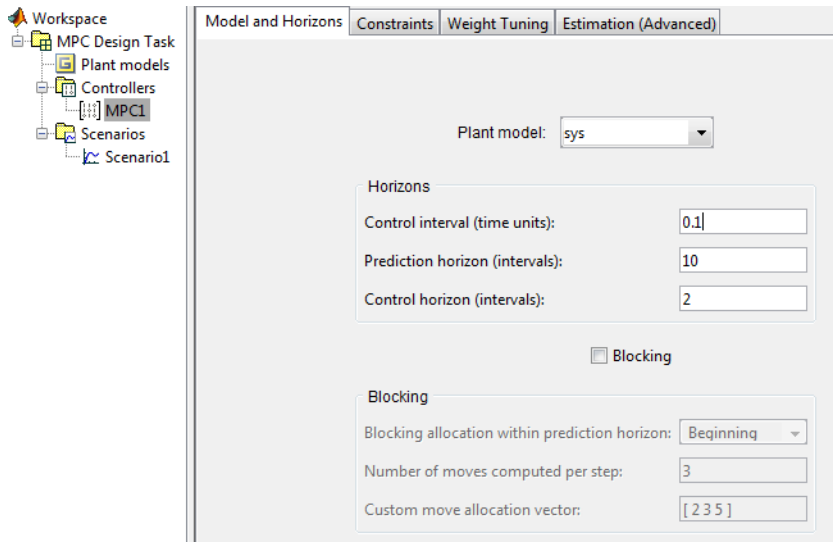
The buttons on the **Controllers** view allow you to:

- **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.

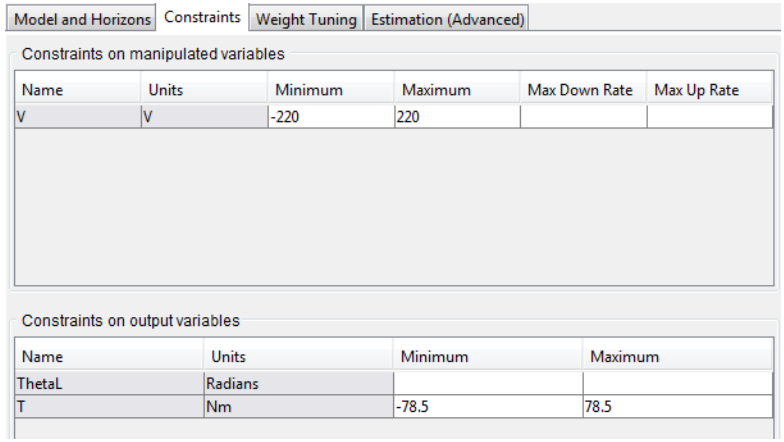
If the selected **Prediction model** is continuous-time, as in this example, the **Control interval** (sampling period) defaults to 1. You need to change this to an application-appropriate value. Set it to 0.1 seconds (as shown in Controller Design View, Models and Horizons Pane). Leave the other values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



Controller Design View, Models and Horizons Pane

Specifying Constraints

Next, click the **Constraints** tab. The view shown in Controller Design View, Constraints Pane 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).

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

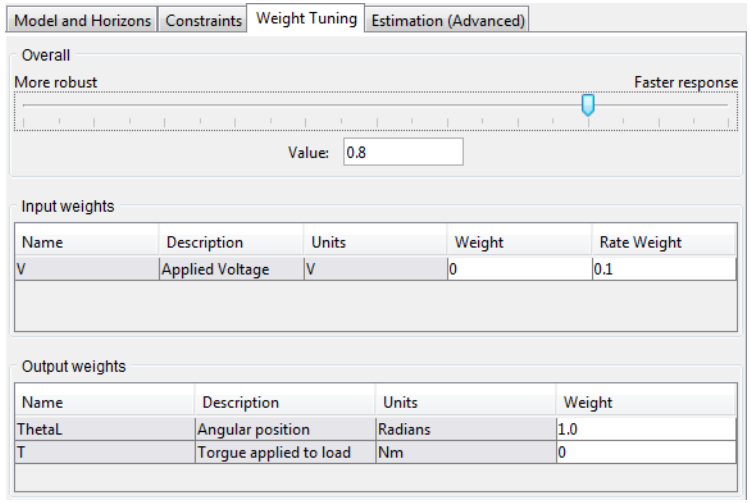
Next, click the **Weight Tuning** tab.

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 2-2 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.

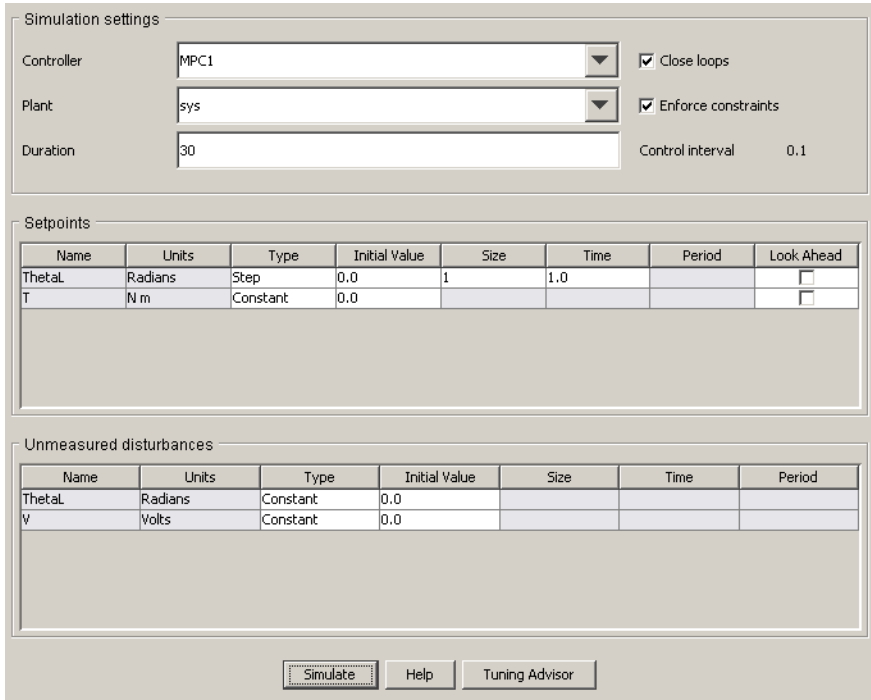
The **Estimation (Advanced)** tab allows you to adjust the controller's response to unmeasured disturbances (not used in this example).

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). Select **Scenario1**.

A *scenario* is a set of simulation conditions. As shown in Simulation Settings View for "Scenario1", 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", 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"

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.

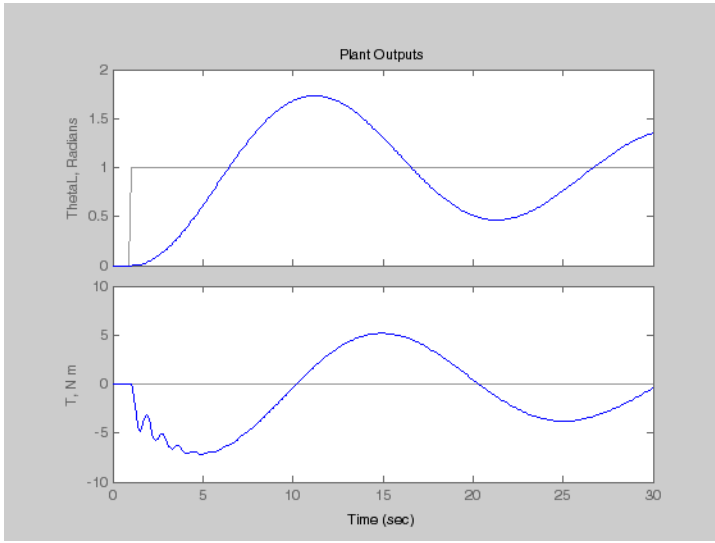
The **Look ahead** option designates that all future setpoint variations are known. In that case, the controller can adjust the manipulated variable(s) in advance to improve setpoint tracking. This would be unusual in practice, and is not being used here.

Running a Simulation

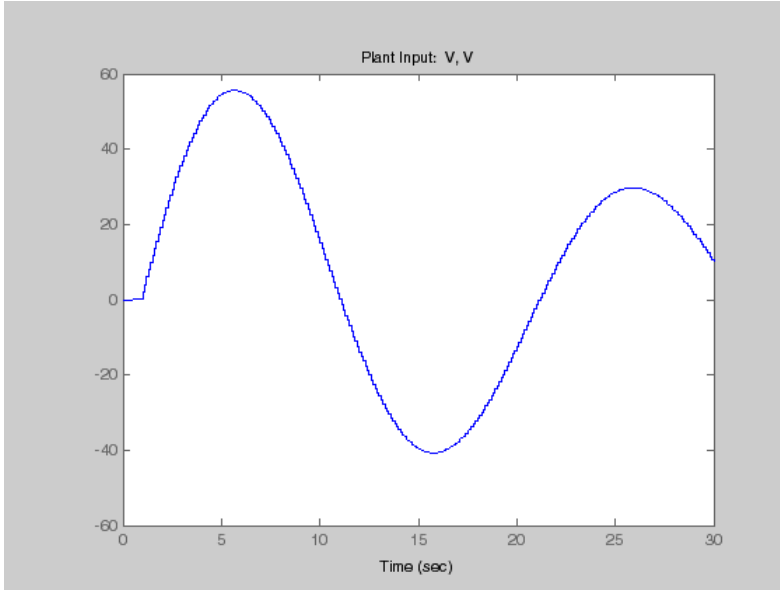
Once you're ready to run the scenario, click the **Simulate** button or the green arrow on the toolbar.

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. 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 provides many of the customization features available in Control System Toolbox `linearSystemAnalyzer` and`controlSystemDesigner` displays. Try clicking a curve to obtain the numerical characteristics of the selected point, or right-clicking in the plot area to open a customization menu.

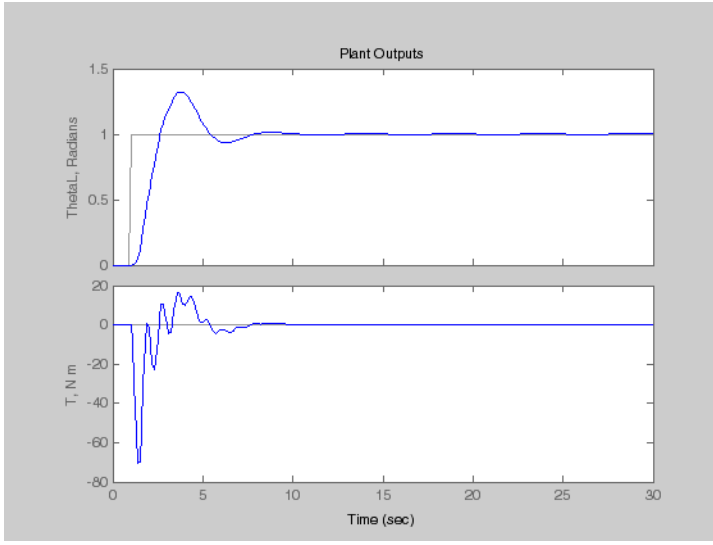
The corresponding applied voltage adjustments appear in a separate window and are also very sluggish.

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

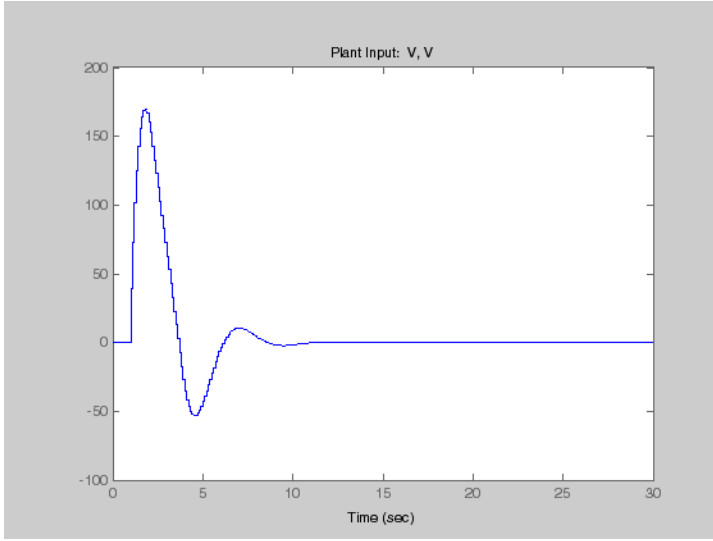
Retuning to Achieve a Faster Servo Response

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

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) and the applied voltage stays within its limits (see Manipulated Variable Adjustments).



Faster Servo Response

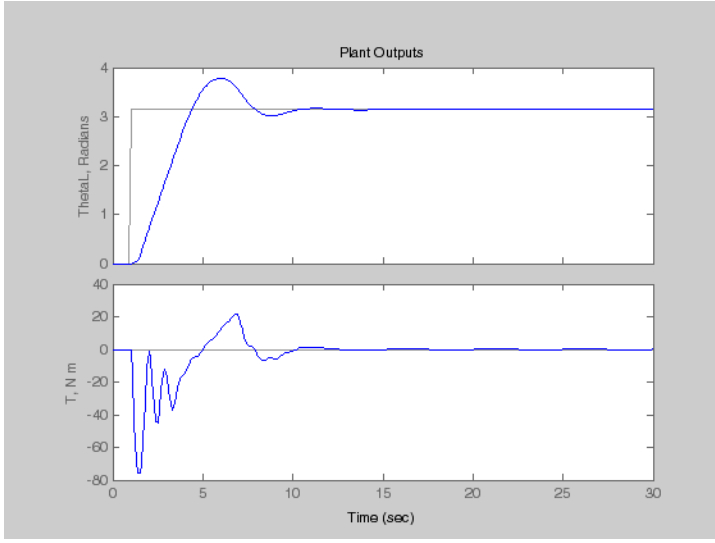


Manipulated Variable Adjustments

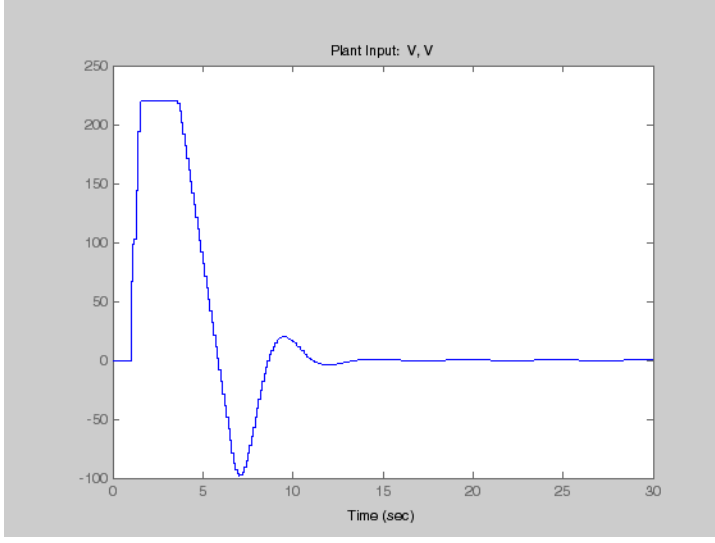
Modifying the Scenario

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

As shown in Servo Response for Step Increase of π Radians and Voltage Adjustments, 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).



Servo Response for Step Increase of π Radians



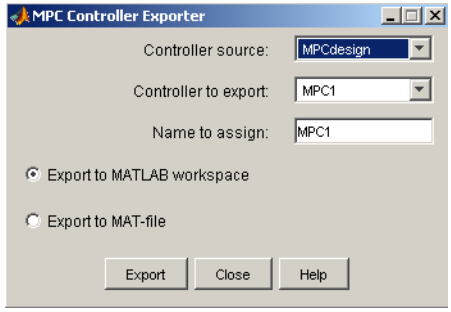
Voltage Adjustments

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 will appear.

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



Exporting a Controller to the Workspace

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

Using Model Predictive Control Toolbox Commands

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

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

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

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

Constructing an MPC Object

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

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

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

Setting, Getting, and Displaying Object Properties

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

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

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

```
get(ServoMPC)
```

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

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

You can also set multiple properties simultaneously.

Set the following properties before continuing with this example:

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

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

```
display(ServoMPC)
```

Running a Simulation

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

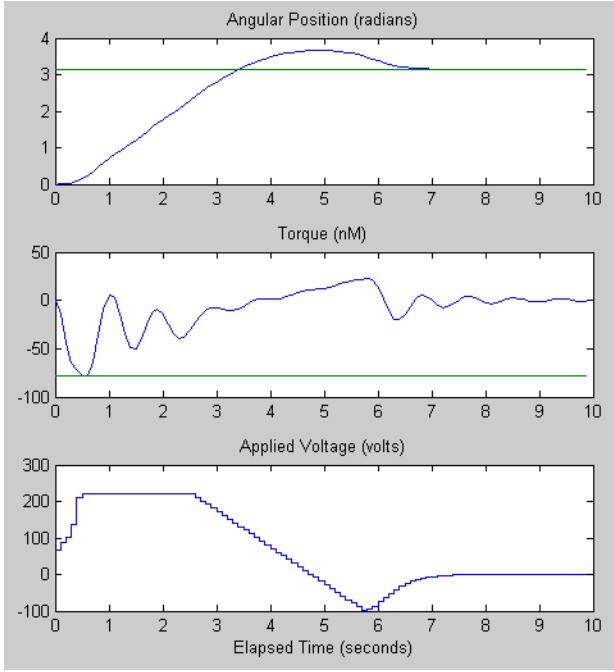
```
TimeSteps = round(10/Ts);
r = [pi 0];
[y, t, u] = sim(ServoMPC, TimeSteps, r);
```

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

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

```
subplot(311)
plot(t, y(:,1), [0 t(end)], pi*[1 1])
title('Angular Position (radians)');
subplot(312)
plot(t, y(:,2), [0 t(end)], [-78.5 -78.5])
title('Torque (nM)')
subplot(313)
stairs(t, u)
title('Applied Voltage (volts)')
xlabel('Elapsed Time (seconds)')
```

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

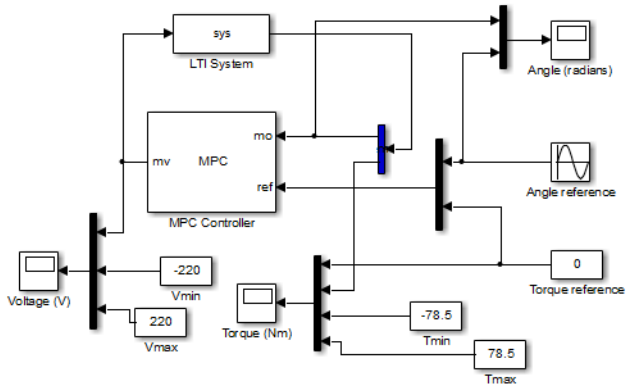


Plotting the Output of the sim Command

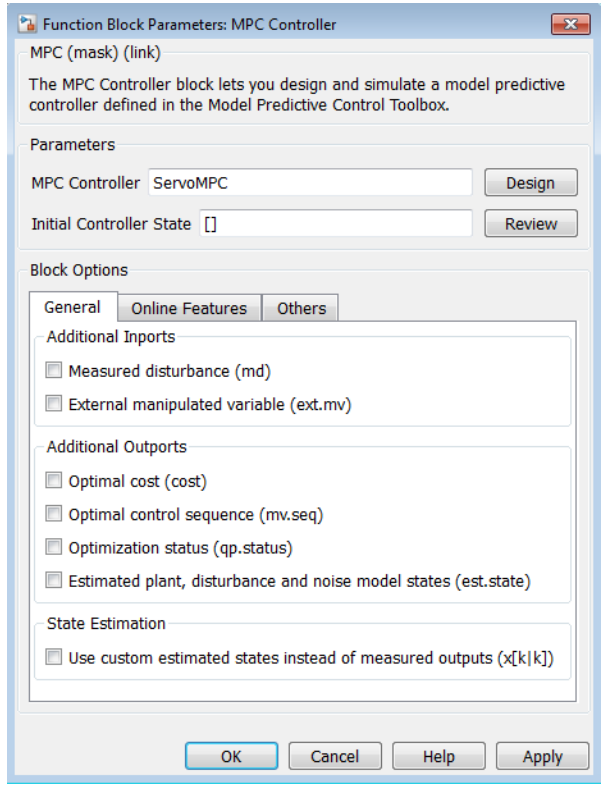
Using MPC Tools in Simulink

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

- Servomechanism Model is an LTI System block from the Control System Toolbox library. The LTI model `sys` (which must exist in the workspace) defines its dynamic behavior. To review how to create this model, see "Defining the Plant Model" on page 4-4.
- MPC Controller is from the MPC Blocks library. Model Predictive Control Toolbox Simulink Block Dialog Box shows the dialog box obtained by double-clicking this block. You need to supply an MPC object, and `ServoMPC` is being used here. It must be in the workspace before you run a simulation. The **Design** button opens the design tool, which allows you to create or modify the object. To review how to use commands to create `ServoMPC`, see "Constructing an MPC Object" on page 4-20.



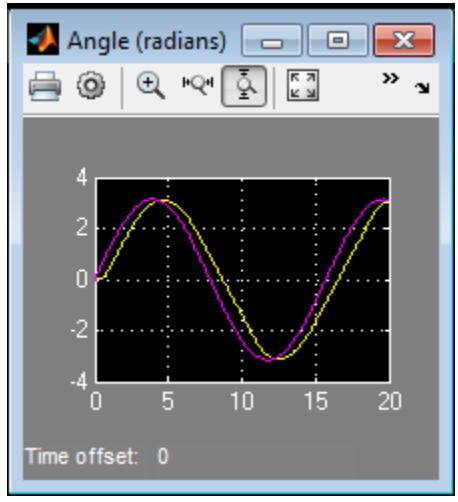
Block Diagram for the Servomechanism Example

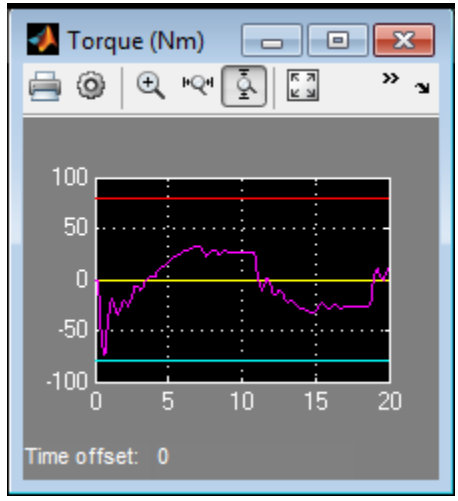


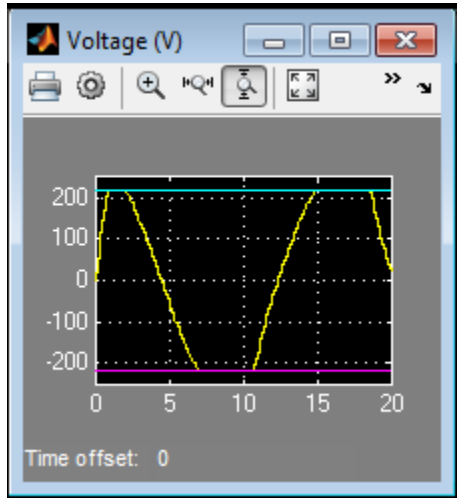
Model Predictive Control Toolbox Simulink Block Dialog Box

The key features of the diagram are as follows:

- The MPC Controller output is the plant input. The Voltage Scope block plots it (yellow curve). Minimum and maximum voltage values are shown as magenta and cyan curves.
- The plant output is a vector signal. The first element is the measured angular position. The second is the unmeasured torque. A Demux block separates them. The angular position feeds back to the controller and plots on the Angle scope (yellow curve). The torque plots on the Torque scope (with its lower and upper bounds).
- The position setpoint varies sinusoidally with amplitude π radians and frequency 0.4 rad/s. It also appears on the Angle scope (magenta curve).


Angle (radians)
4
2
0
-2
-4
0
5
10
15
20
Time offset: 0


Torque (Nm)
100
50
0
-50
-100
0
5
10
15
20
Time offset: 0



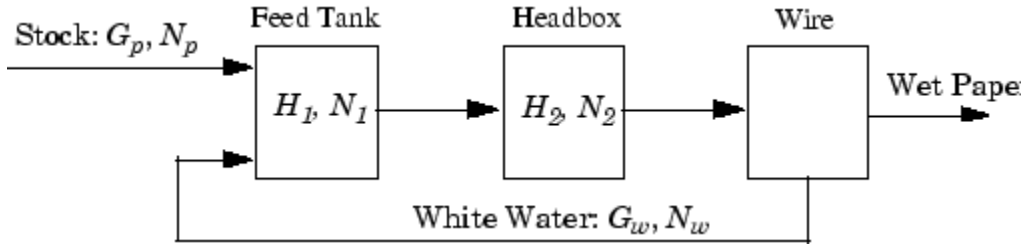
The angular position tracks the sinusoidal setpoint variations well despite saturation of the applied voltage. The setpoint variations are more gradual than the step changes used previously, so the torque stays well within its bounds.

Paper Machine Process Control

In this section...

System Model

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



Schematic of Paper Machine Headbox Elements

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

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

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

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

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

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

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

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

$v = N_p$

The white water consistency is an unmeasured disturbance.

$d = N_w$

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

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

Linearizing the Nonlinear Model

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

```
A = [-1.9300         0         0         0
    0.3940   -0.4260         0         0
         0         0   -0.6300         0
    0.8200   -0.7840    0.4130   -0.4260];
B = [1.2740    1.2740         0         0
         0         0         0         0
    1.3400   -0.6500    0.2030    0.4060
         0         0         0         0];
C = [0    1.0000         0         0
         0         0    1.0000         0
```

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

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

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

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

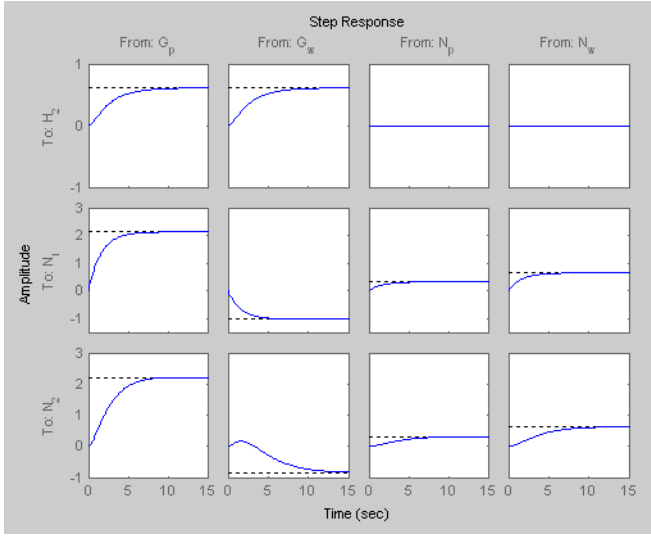
As a quick check of model validity, plot its step responses as follows:

```
step(PaperMach);
```

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

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

These features make it difficult to achieve accurate, independent control of H_2 and N_2.



Linearized Paper Machine Model's Step Responses

MPC Design

Type

```
mpctool
```

to open the MPC design tool. Import your LTI Paper Mach model as described in "Opening MPCTOOL and Importing a Model" on page 4-5.

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.

Input signal properties

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

Output signal properties

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

Signal Properties for the Paper Machine Application

Initial Controller Design

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

- **Models and Horizons**. Control interval = 2 minutes
- **Constraints**. For both G_p and G_w, Minimum = –10, Maximum = 10, Max down rate = –2, Max up rate = 2.
- **Weight Tuning**. For both G_p and G_w, Weight = 0, Rate weight = 0.4.

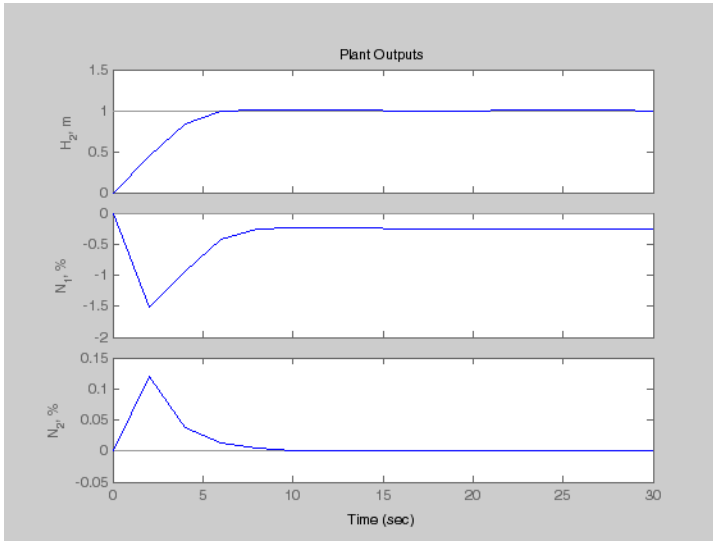
For N_1, Weight = 0. (Other outputs have Weight = 1.)

Servo Response

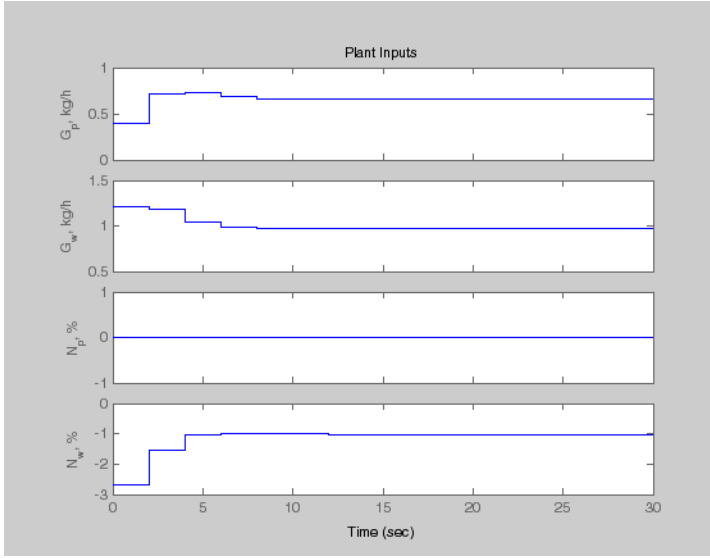
Finally, select the **Scenario1** node and define a servo-response test:

- Duration = 30
- H_2 setpoint = 1 (constant)

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



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). 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 4-30). 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).

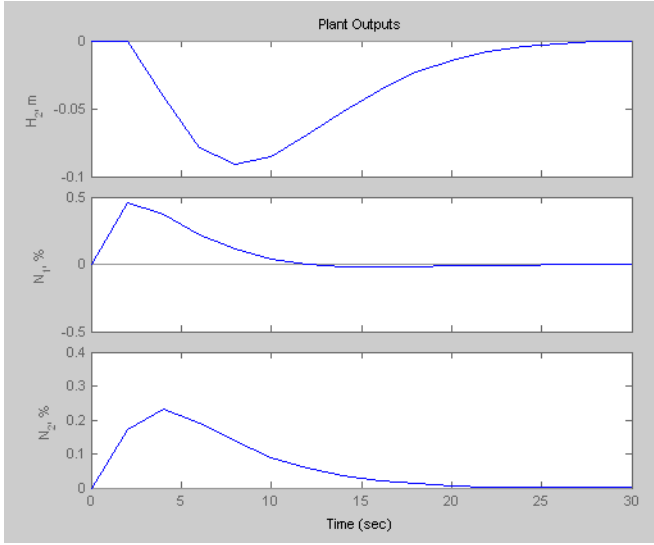
Experiment with the "look ahead" feature. First, observe that in the simulation just completed the manipulated variables didn't begin to move until the disturbance occurred at t = 10 minutes. Return to the **Feedforward** scenario, select the **Look ahead** option for the measured disturbance, and repeat 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. The two controlled outputs (H_2 and N_2) exhibit relatively small deviations from their setpoints (which are zero). The settling time is longer than for the servo response (compare to Servo Response for Unit Step in Headbox Level Setpoint) which is typical.



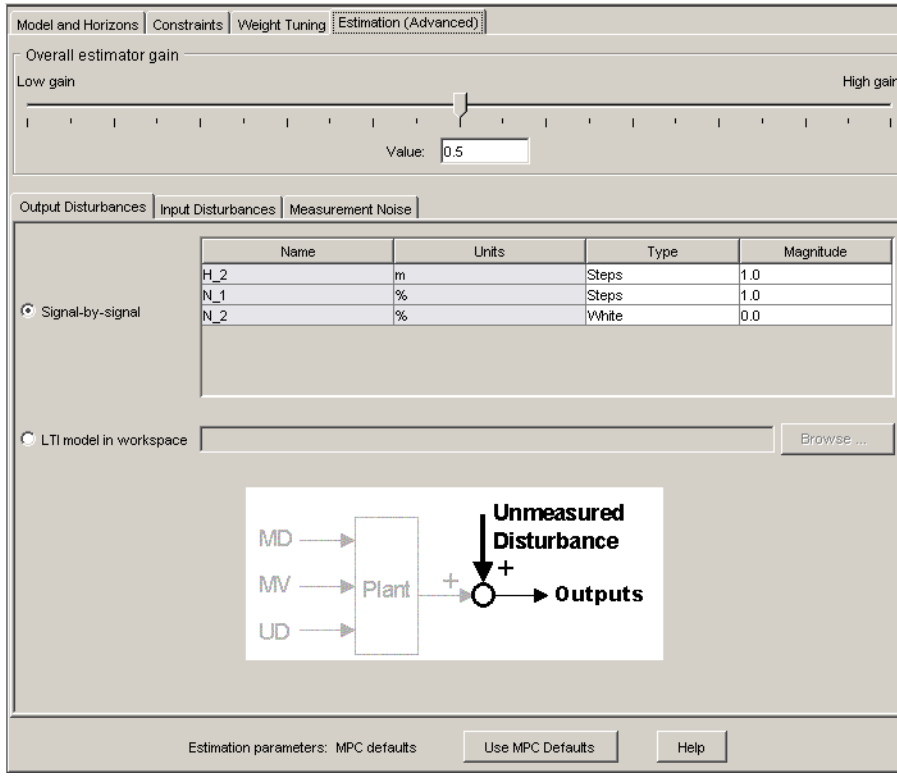
Feedback Scenario: Unmeasured Disturbance Rejection

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 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. 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.

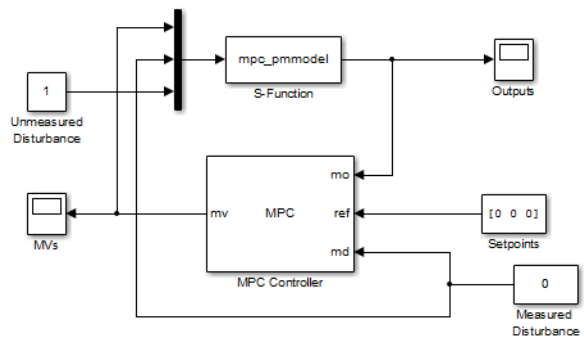
In the paper machine application, the default disturbance assumptions are reasonable. It is difficult to improve disturbance rejection significantly by modifying them.

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 4-33. Such tests don't introduce prediction error, and are a useful benchmark for more demanding tests with a nonlinear plant model. The controller's prediction model is linear, so such tests introduce prediction error.

Open the paper machine headbox control Simulink model by typing:

```
mpc_papermachine
```

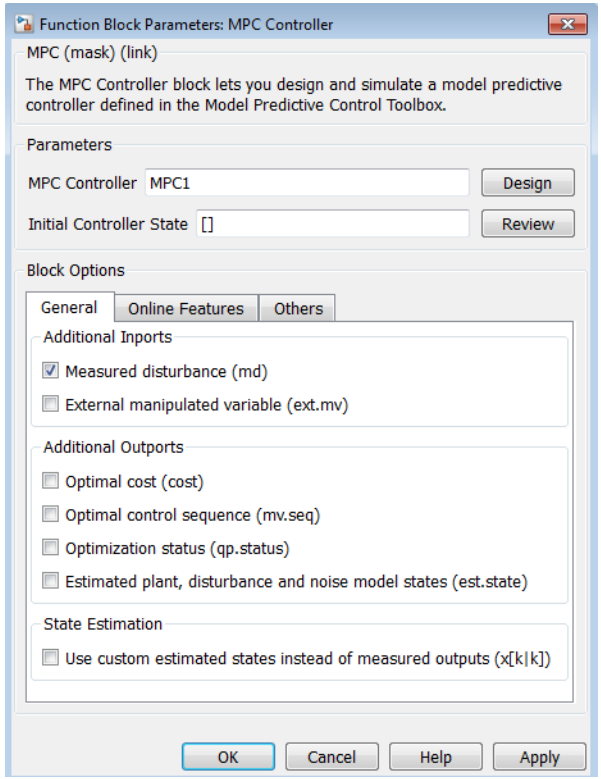


Paper Machine Headbox Control Using MPC Tools in Simulink

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

As shown in the following dialog box, the MPC block references a controller design called `MPC1`, which was exported to the MATLAB workspace from the design tool. Note also

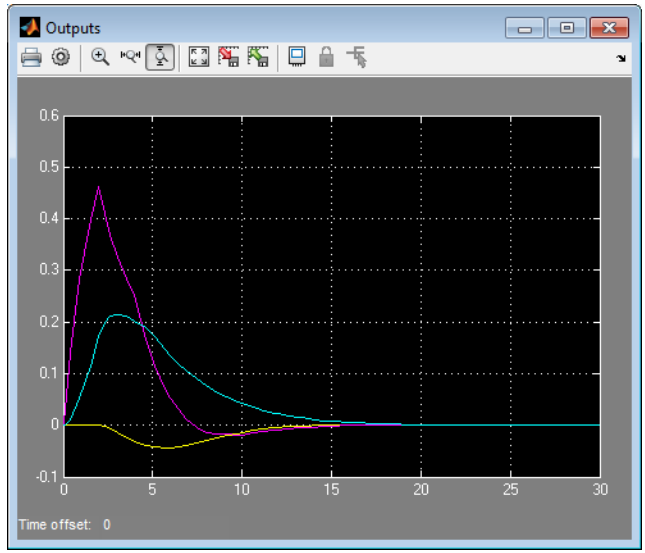
that the measured disturbance inport is enabled, allowing the measured disturbance to be connected as shown in Paper Machine Headbox Control Using MPC Tools in Simulink.



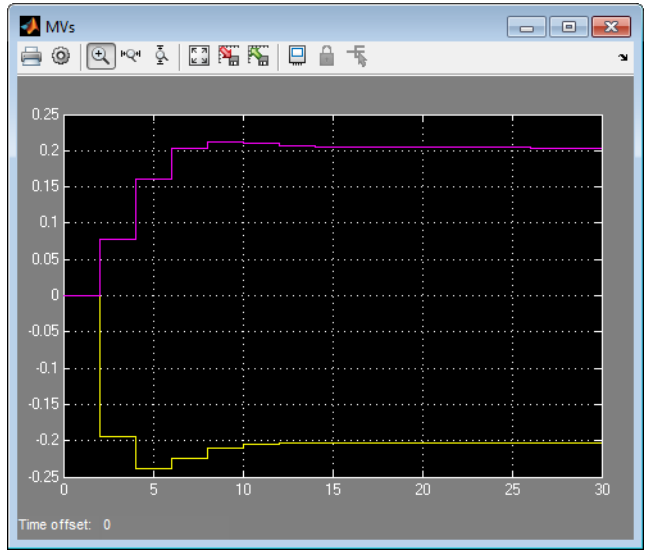
Test, Output Variables shows the scope display from the "Outputs" block for the setup of Paper Machine Headbox Control Using MPC Tools in Simulink, i.e., an unmeasured disturbance. The yellow curve is H_2, the magenta is N_1, and the cyan is N_2. Comparing to Feedback Scenario: Unmeasured Disturbance Rejection, the results are almost identical, indicating that the effects of nonlinearity and prediction error were insignificant in this case. Simulink Test, Manipulated Variables shows the corresponding manipulated variable moves (from the "MVs" scope in Paper Machine Headbox Control Using MPC Tools in Simulink) 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 and Simulink Test, Manipulated Variables (scaled by a factor of 4), but for a disturbance size

of 6, the setpoint deviations are relatively larger, and the curve shapes differ (not shown). There are marked qualitative and quantitative differences when the disturbance size is 8. When it is 9, deviations become very large, and the MVs saturate. If such disturbances were likely, the controller would have to be retuned to accommodate them.



Test, Output Variables



Simulink Test, Manipulated Variables

References

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

Bumpless Transfer Between Manual and Automatic Operations

In this section...

This example shows how to bumplessly transfer between manual and automatic operations of a plant.

During startup of a manufacturing process, operators adjust key actuators manually until the plant is near the desired operating point before switching to automatic control. If not done correctly, the transfer can cause a *bump*, that is, large actuator movement.

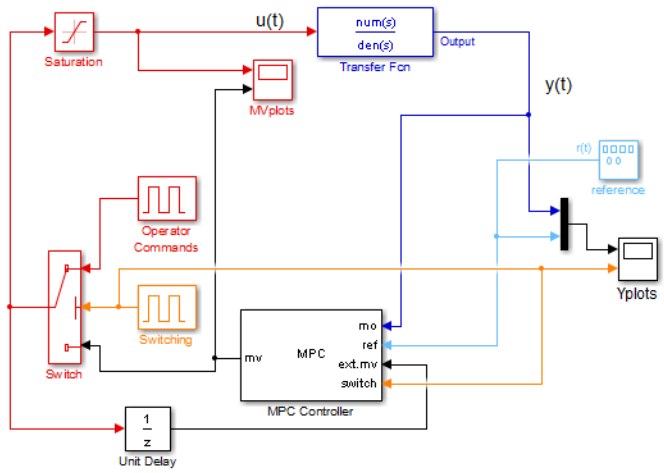
In this example, you simulate a Simulink model that contains a single-input single-output LTI plant and an MPC Controller block.

A model predictive controller monitors all known plant signals, even when it is not in control of the actuators. This monitoring improves its state estimates and allows a bumpless transfer to automatic operation.

Open Simulink Model

Open the Simulink model.

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



To simulate switching between manual and automatic operation, the Switching block sends either 1 or 0 to control a switch. When it sends 0, the system is in automatic mode, and the output from the MPC Controller block 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 `ext.mv` inport, unless the plant input saturates at –1 or 1. The controller constantly monitors the plant output and updates its estimate of the plant state, even when in manual operation.

This model also shows the optimization switching option. When the system switches to manual operation, a nonzero signal enters the `switch` inport of the controller block. The signal turns off the optimization calculations, which reduces computational effort.

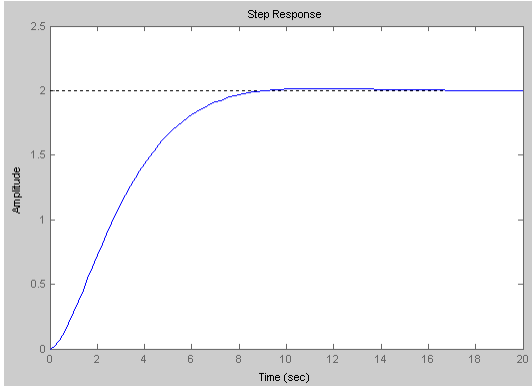
Define Plant and MPC Controller

Create the plant model.

```
num=[1 1];
den=[1 3 2 0.5];
sys=tf(num,den);
```

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

```
step(sys)
```



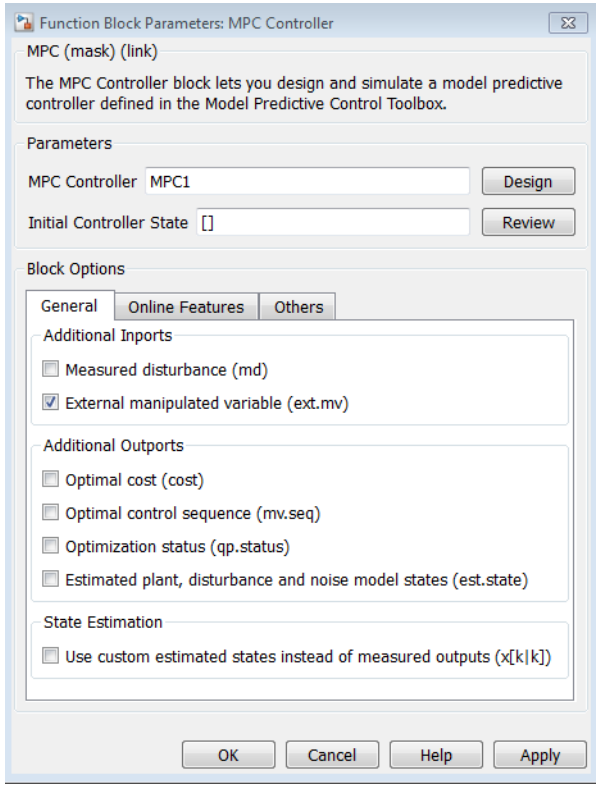
Create an MPC controller.

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

Configure MPC Block Settings

Open the Function Block Parameters: MPC Controller dialog box.

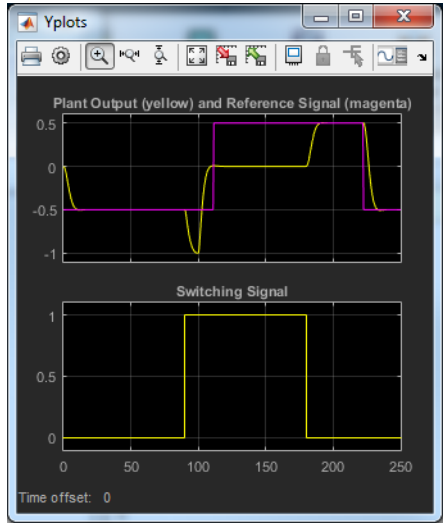
- Specify `MPC1` in the **MPC Controller** box.
- Verify that the **External Manipulated Variable (ext.mv)** option in the **General** tab is selected. This option adds the `ext.mv` inport to the block to enable the use of external manipulated variables.
- Verify that the **Use external signal to enable or disable optimization (switch)** option in the **Others** tab is selected. This option adds the `switch` inport to the controller block to enable switching off the optimization calculations.

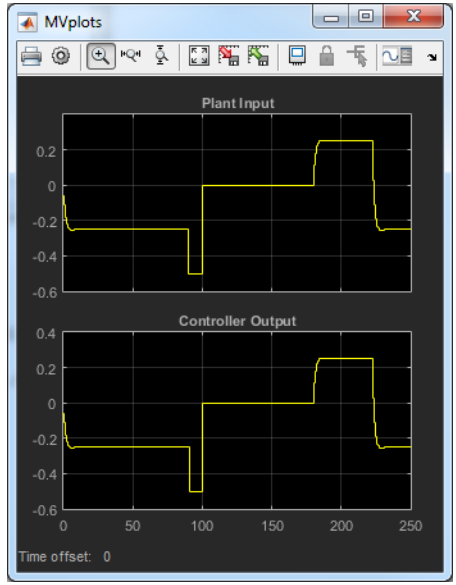


Click **OK**.

Examine Switching Between Manual and Automatic Operation

Click **Run** in the Simulink model window to simulate the model.





For the first 90 time units, the `Switching Signal` is 0, which makes the system operate in automatic mode. During this time, the controller smoothly drives the controlled plant output from its initial value, 0, to the desired reference value, –0.5.

The controller state estimator has zero initial conditions as a default, which is appropriate when this simulation begins. Thus, there is no bump at startup. In general, start the system running in manual mode long enough for the controller to acquire an accurate state estimate before switching to automatic mode.

At time 90, the `Switching Signal` changes to 1. This change switches the system to manual operation and sends the operator commands to the plant. Simultaneously, the nonzero signal entering the `switch` inport of the controller turns off the optimization calculations. While the optimization is turned off, the MPC Controller block passes the current `ext.mv` signal to the `Controller Output`.

Once in manual mode, the operator commands set the manipulated variable to –0.5 for 10 time units, and then to 0. The `Plant Output` plot shows the open-loop response between times 90 and 180 when the controller is deactivated.

At time 180, the system switches back to automatic mode. As a result, the plant output returns to the reference value smoothly, and a similar smooth adjustment occurs in the controller output.

Turn off Manipulated Variable Feedback

Delete the signals entering the `ext.mv` and `switch` inports of the controller block.

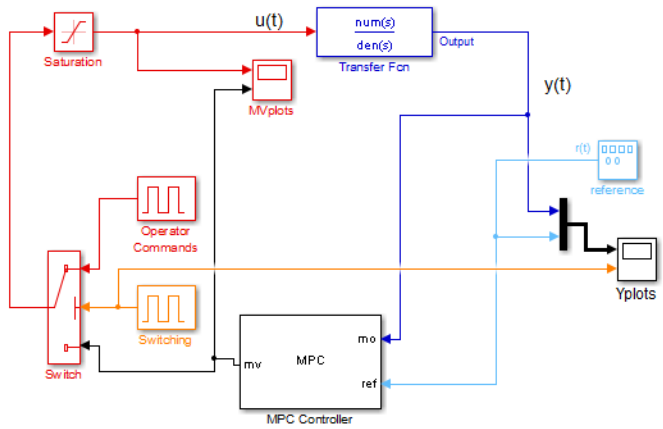
Delete the Unit Delay block and the signal line entering its inport.

Open the Function Block Parameters: MPC Controller dialog box.

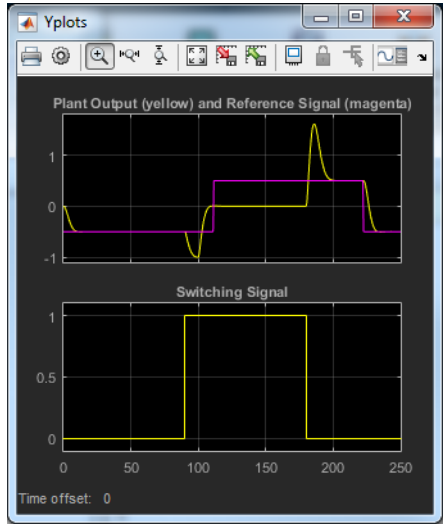
Deselect the **External Manipulated Variable (ext.mv)** option in the **General** tab to remove the `ext.mv` inport from the controller block.

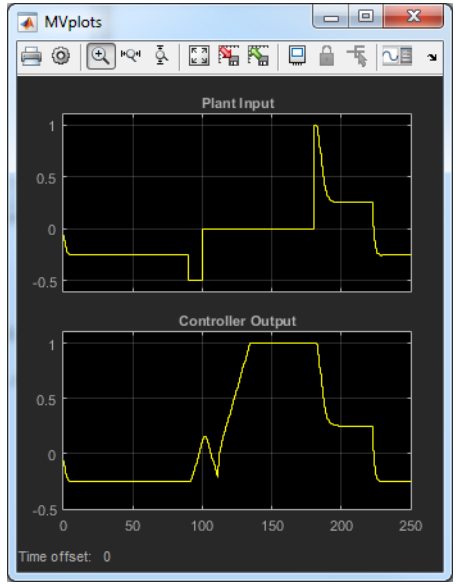
Deselect the **Use external signal to enable or disable optimization (switch)** option in the **Others** tab to remove the `switch` inport from the controller block.

Click **OK**. The Simulink model now resembles the following figure.



Click **Run** to simulate the model.





The behavior is identical to the original case for the first 90 time units.

When the system switches to manual mode at time 90, the plant behavior is the same as before. However, the controller tries to hold the plant at the setpoint. So, its output increases and eventually saturates, as seen in `Controller Output`. Since the controller assumes that this output is going to the plant, its state estimates become inaccurate. Therefore, when the system switches back to automatic mode at time 180, there is a large bump in the `Plant Output`.

Such a bump creates large actuator movements within the plant. By smoothly transferring from manual to automatic operation, a model predictive controller eliminates such undesired movements.

Related Examples

- “Switching Controller Online and Offline with Bumpless Transfer” on page 4-48

Switching Controller Online and Offline with Bumpless Transfer

This example shows how to obtain bumpless transfer when switching model predictive controller from manual to automatic operation or vice versa.

In particular, it shows how the EXT.MV input signal to the MPC block can be used to keep the internal MPC state up to date when the operator or another controller is in control.

Define Plant Model

The linear open-loop dynamic plant model is as follows:

```
num = [1 1];
den = [1 3 2 0.5];
sys = tf(num,den);
```

Design MPC Controller

Construct MPC controller

Create an MPC controller with plant model, sample time and horizons.

```
Ts = 0.5;           % Sampling time
p = 15;             % Prediction horizon
m = 2;              % Control horizon
mpcobj = mpc(sys,Ts,p,m);
```

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

MV Constraints

Define constraints on the manipulated variable.

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

Weights

Change the output weight.

```
mpcobj.Weights.Output=0.01;
```

Simulate Using Simulink®

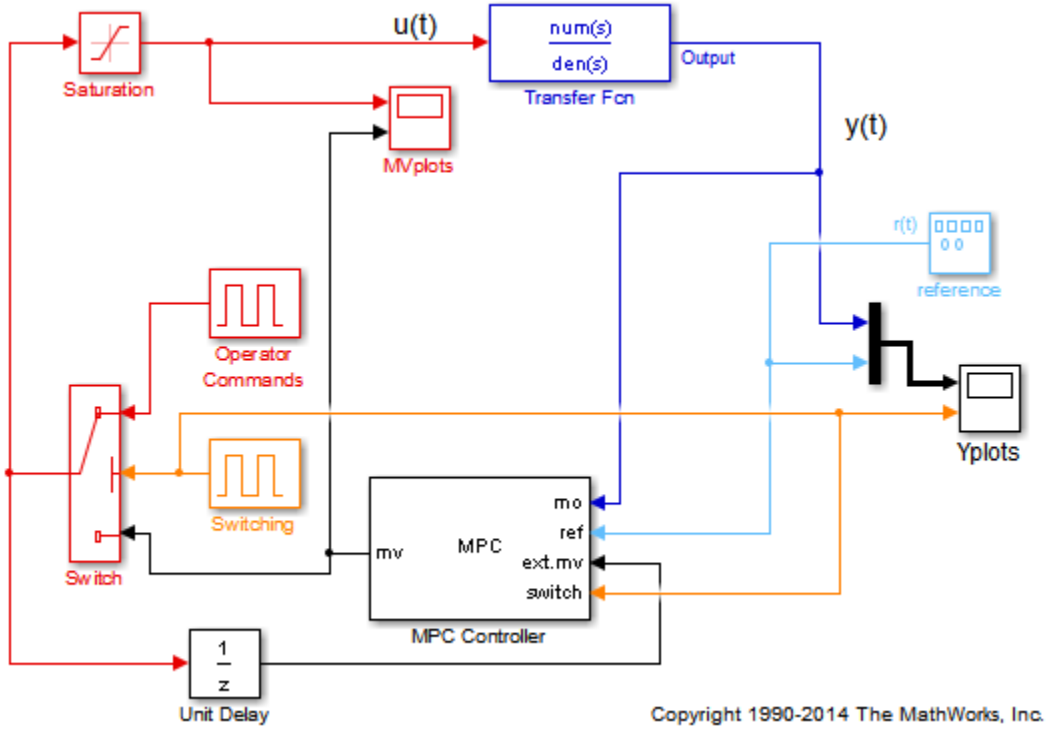
To run this example, Simulink® is required.

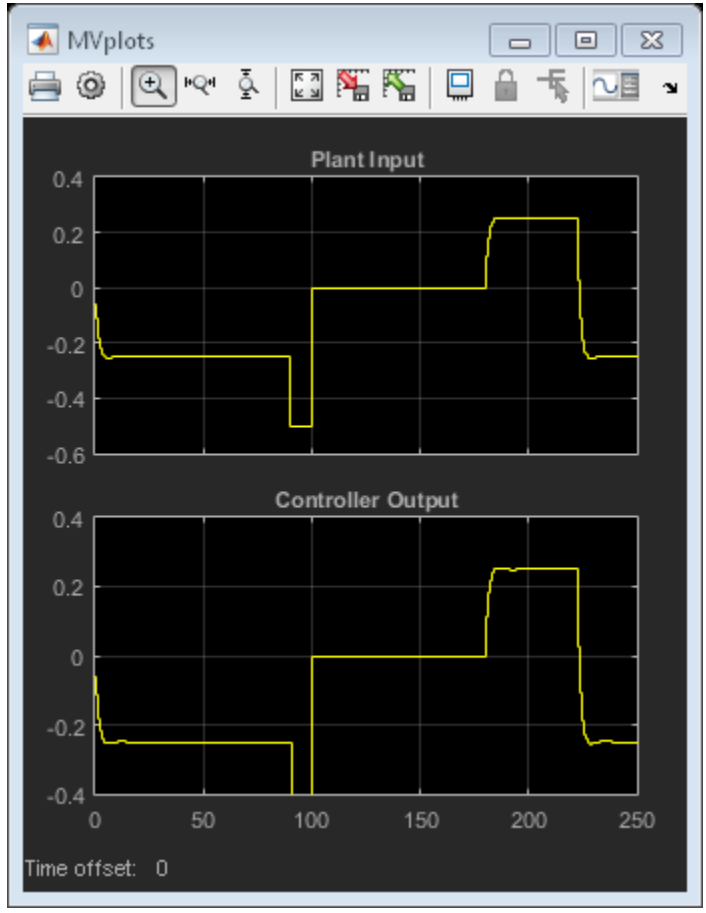
```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
```

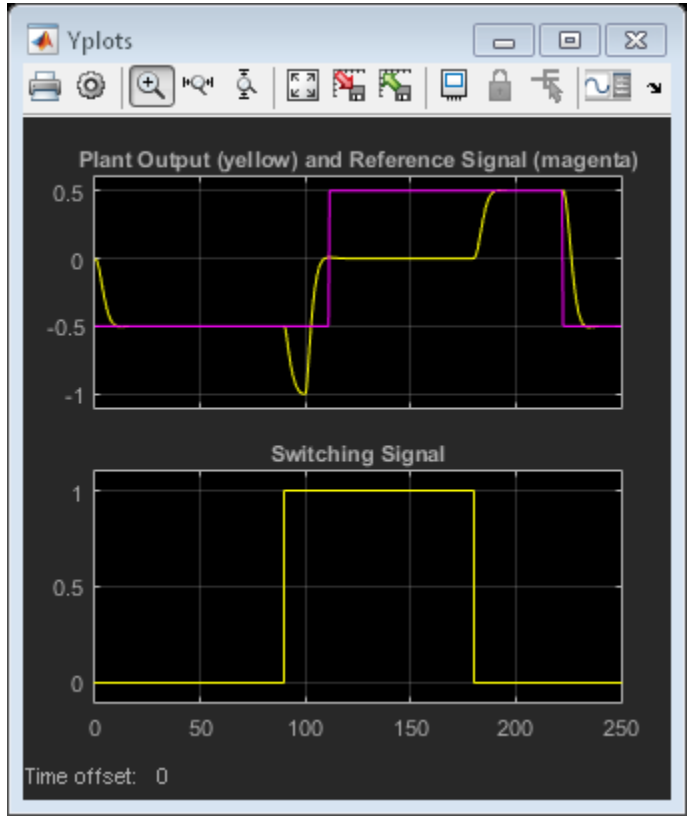
Simulate closed-loop control of the linear plant model in Simulink. Controller "mpcobj" is specified in the block dialog.

```
mdl = 'mpc_bumpless';
open_system(mdl)
sim(mdl)
```

```
-->Converting the "Model.Plant" property of "mpc" object to state-space.
-->Converting model to discrete time.
-->Integrated white noise added on measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```




MVplots
Plant Input
0.4
0.2
0
-0.2
-0.4
-0.6
Controller Output
0.4
0.2
0
-0.2
-0.4
0
50
100
150
200
250
Time offset: 0

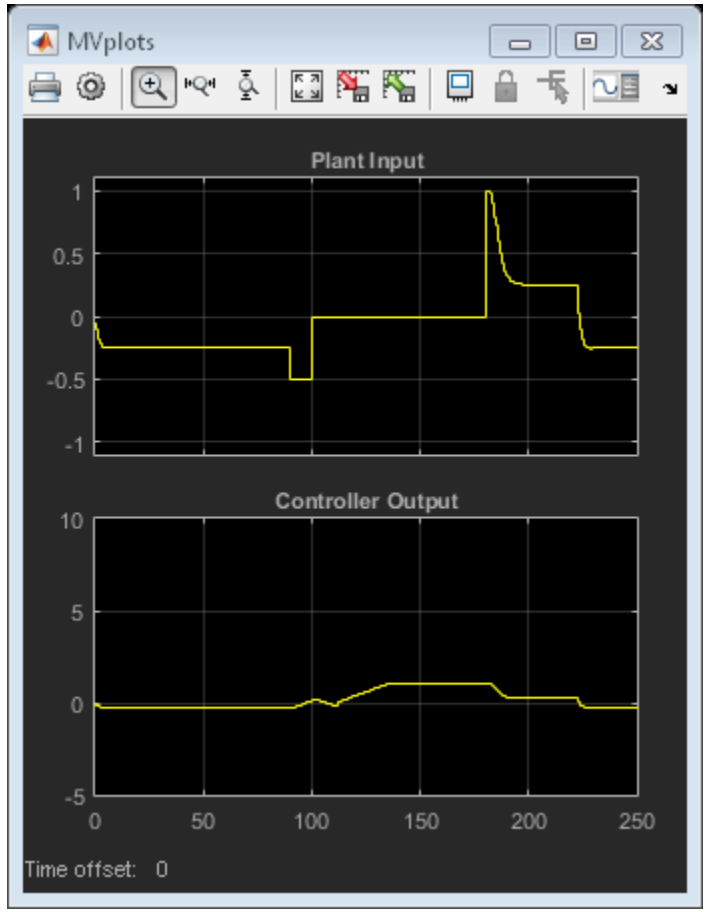


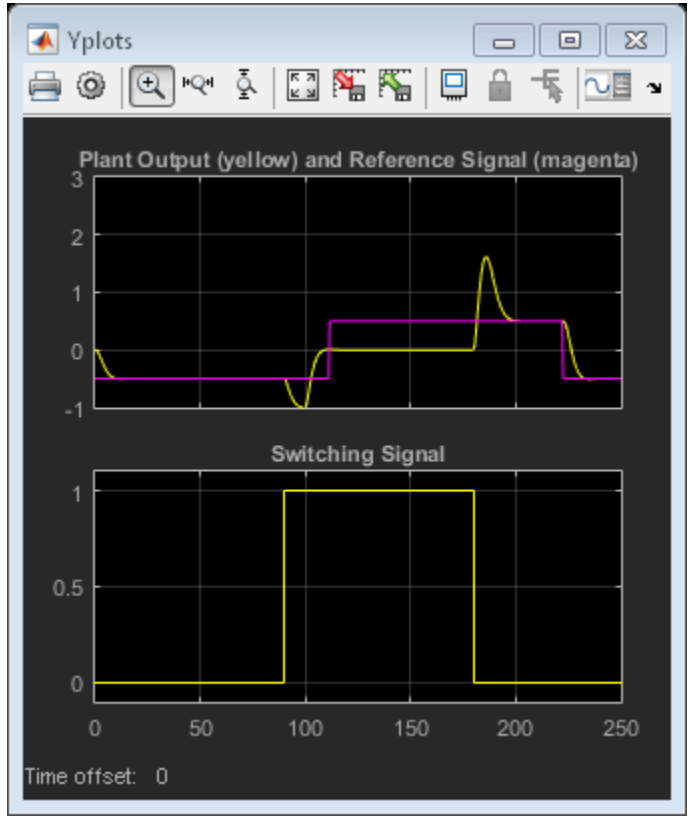
Simulate without Using External MV Signal

Without using the external MV signal, MPC controller is no longer able to provide bumpless transfer because the internal controller states are not estimated correctly.

```
delete_line(mdl,'Switch/1','Unit Delay/1');
delete_line(mdl,'Unit Delay/1','MPC Controller/3');
delete_block([mdl '/Unit Delay']);
delete_line(mdl,'Switching/1','MPC Controller/4');
set_param([mdl '/MPC Controller'],'mv_inport','off');
set_param([mdl '/MPC Controller'],'switch_inport','off');
set_param([mdl '/Yplots'],'Ymin','-1~-0.1')
set_param([mdl '/Yplots'],'Ymax','3~1.1')
set_param([mdl '/MVplots'],'Ymin','-1.1~-5')
set_param([mdl '/MVplots'],'Ymax','1.1~10')
```

```
sim(mdl);
```


MVplots
Plant Input
1
0.5
0
-0.5
-1
Controller Output
10
5
0
-5
0
50
100
150
200
250
Time offset: 0



Now the transition from manual to automatic control is much less smooth. Note the large "bump" between time = 180 and 200.

```
bdclose(mdl)
```

Related Examples

- “Bumpless Transfer Between Manual and Automatic Operations” on page 4-40

Coordinate Multiple Controllers at Different Operating Points

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

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

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

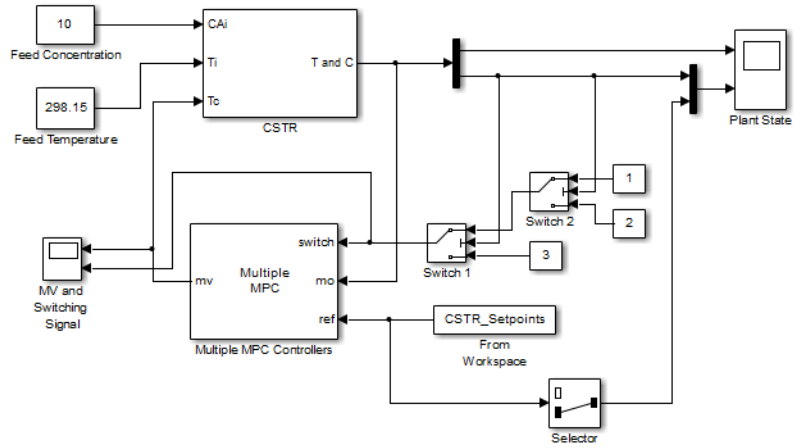
The subject process is a constant-volume continuous stirred-tank reactor (CSTR). The model consists of two nonlinear ordinary differential equations (see [1]). The model states are the reactor temperature and the rate-limiting reactant concentration. For the purposes of this example, both are assumed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

There are three inputs:

- Concentration of the limiting reactant in the reactor feed stream, $kmol/m^3$
- The reactor feed temperature, K
- The coolant temperature, K

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

The Simulink diagram for this example appears below. The CSTR model is a masked subsystem. The feed temperature and composition are constants. As discussed above, the control system adjusts the coolant temperature (the `Tc` input on the CSTR block).



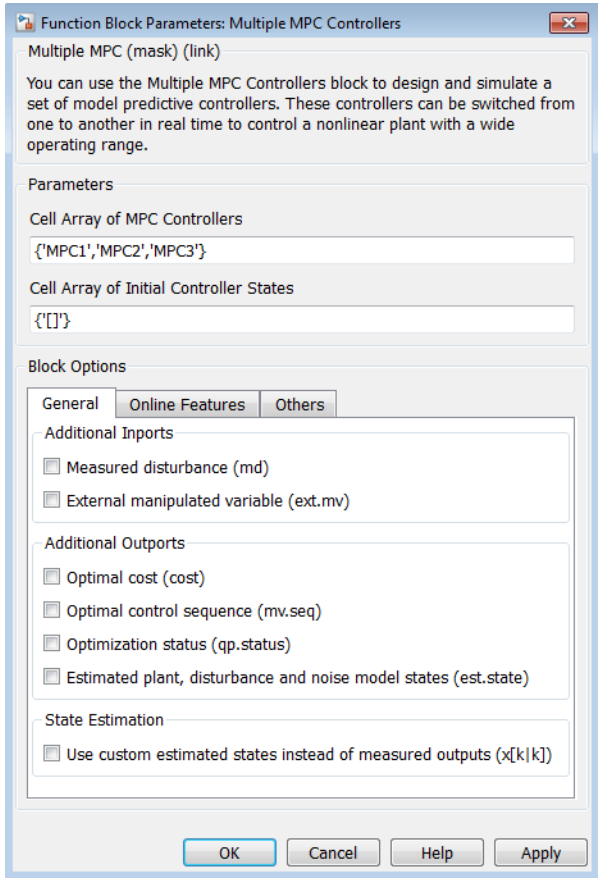
The two CSTR outputs are the reactor temperature and composition respectively. These are being sent to a scope display and to the control system as feedback.

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

In that case why supply the temperature measurement to the controller? The main reason is to improve state estimation. If this were not done, the control system would have to infer the temperature value from the concentration measurement, which would introduce an estimation error and degrade the model's predictive accuracy.

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

The figure below shows the `Multi MPC Controller` mask. The block is coordinating three controllers (`MPC1`, `MPC2` and `MPC3` in that sequence). It is also receiving the setpoint signal from the workspace, and the **Look ahead** option is active. This allows the controller to anticipate future setpoint values and usually improves setpoint tracking.



In order to designate which one of the three controllers is active at each time instant, we send the `Multi MPC Controllers` block a switching signal (connected to its `switch` input port). If it is 1, `MPC1` is active. If it is 2, `MPC2` is active, and so on.

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

- If the reactor concentration is 8 kmol/m^3 or greater, `Switch 1` sends the constant 1 to its output. Otherwise it sends the constant 2.
- If the reactor concentration is 3 kmol/m^3 or greater, `Switch 2` passes through the signal coming from `Switch 1` (either 1 or 2). Otherwise is sends the constant 3.

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

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

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

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

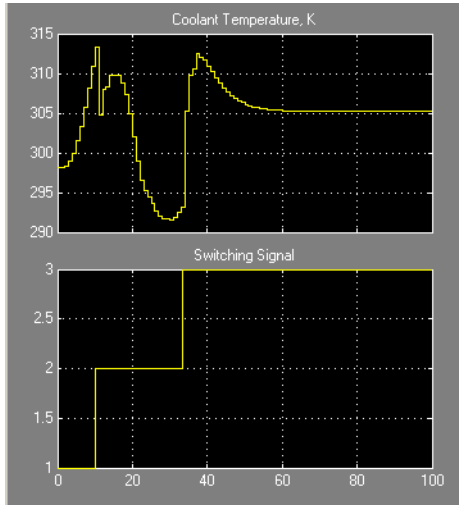
The following code obtains the linear models and designs the three controllers

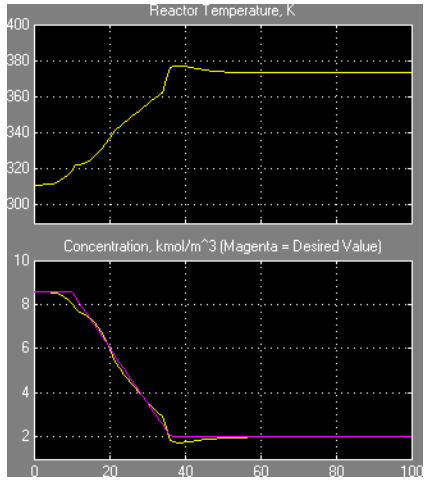
```
[sys, xp] = CSTR_INOUT([],[],[],'sizes');
up = [10 298.15 298.15]';
yp = xp;
Ts = 1;
Nc = 3;
Controllers = cell(1,3);
Concentrations = [8.5 5.5 2];
Y = yp;
for i = 1:Nc
    clear Model
    Y(2) = Concentrations(i);
    [X,U,Y,DX]=trim('CSTR_INOUT',xp(:),up(:),Y(:),[],[1,2]',2)
 [a,b,c,d]=linmod('CSTR_INOUT', X, U );
    Plant = ss(a,b,c,d);
    Plant.InputGroup.MV = 3;
    Plant.InputGroup.UD = [1,2];
    Model.Plant = Plant;
 Model.Nominal.U = [0; 0; up(3)];
```

```
    Model.Nominal.X = xp;
    Model.Nominal.Y = yp;
    MPCobj = mpc(Model, Ts);
    MPCobj.Weight.OV = [0 1];
    D = ss(getindist(MPCobj));
    D.b = D.b*10;
    set(D,'InputName',[],'OutputName',[],'InputGroup',[], ...
  'OutputGroup',[]);
    setindist(MPCobj, 'model', D);
    Controllers{i} = MPCobj;
end
MPC1 = Controllers{1};
MPC2 = Controllers{2};
MPC3 = Controllers{3}
```

The key points regarding the designs are as follows:

- All three controllers use the same nominal condition, the values of the plant inputs and outputs at the initial steady-state. Exception: all unmeasured disturbance inputs must have zero nominal values.
- Each controller employs a different prediction model. The model structure is the same in each case (input and outputs are identical in number and type) but each model represents a particular steady-state reactor composition.
- It turns out that the MPC2 plant model obtained at 5 kmol/m^3 is open-loop unstable. We must use a model structure that promotes a stable Kalman state estimator. If we include the unmeasured disturbance inputs in the prediction model, the default estimator assumes integrated white noise at each such input, which produces a stable estimator in this case.
- The default estimator signal-to-noise settings are inappropriate, however. If you use them and monitor the state estimates (not shown), the internally estimated temperature and composition can be far from the measured values. To overcome this, we increase the signal-to-noise ratio in each disturbance channel. See the use of `getindist` and `setindist` above. The default signal to noise is being increased by a factor of 10.
- We are using a zero weight on the measured temperature. See the above discussion of control objectives for the rationale.



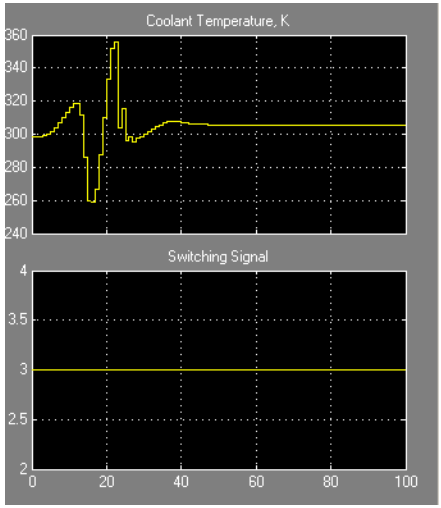


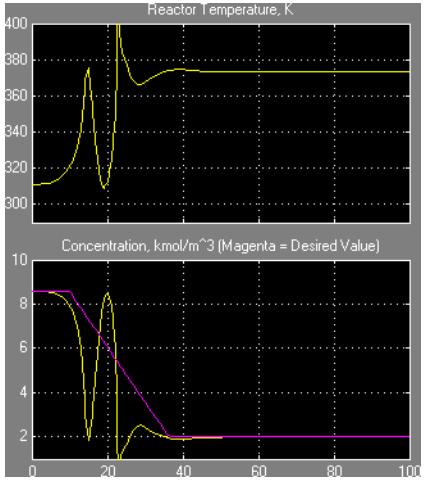
The above plots show the simulation results. The Multi MPC Controller block uses the three controllers sequentially as expected (see the switching signal). Tracking of the concentration setpoint is excellent and the reactor temperature is also controlled well.

To achieve this, the control system starts by increasing the coolant temperature, causing the reaction rate to increase. Once the reaction has achieved a high rate, it generates substantial heat and the coolant temperature must decrease to keep the

reactor temperature under control. As the reactor concentration depletes, the reaction rate slows and the control system must again raise the coolant temperature, finally settling at 305 K, about 7 K above the initial condition.

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





Using Custom Constraints in Blending Process

In this section...

About the Blending Process

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

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

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

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

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

At the nominal operating condition:

- Feed 1 ϕ_1 (mostly constituent 1) is 80% of the total inflow ϕ.
- Feed 2 ϕ_2 (mostly constituent 2) is 20%.
- Feed 3 ϕ_3 (pure constituent 1) is not used.

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

$\phi_1 = \phi_T - \phi_2 - \phi_3$

Each feed has limited availability:

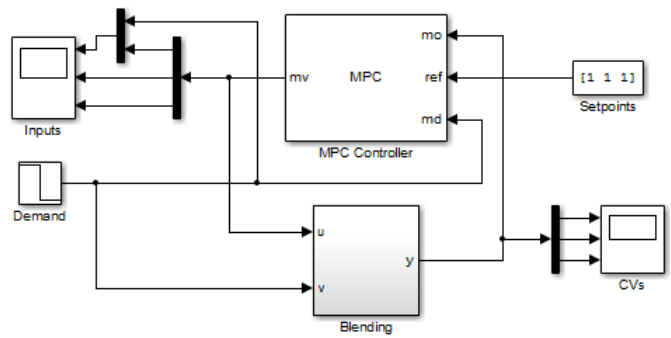
$0 \le \phi_i \le \phi_{i,max}$

The equations are normalized such that—at the nominal steady state—the mean residence time in the mixing container is $\tau = 1$. The target inventory is $V = 1$, and the target blend composition is $\gamma_1 = \gamma_2 = 1$.

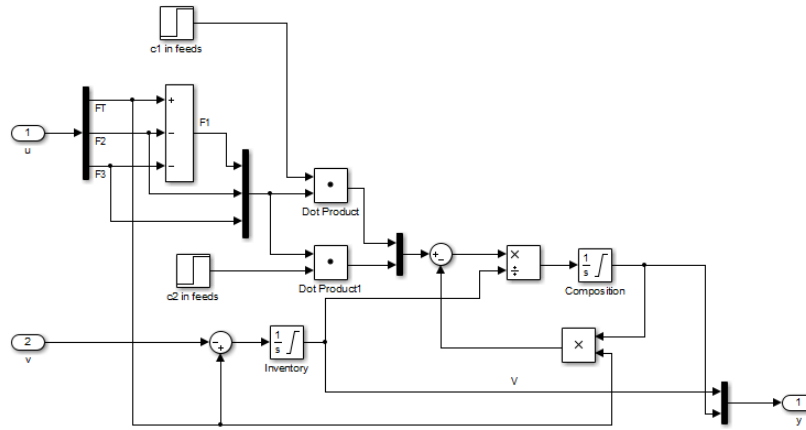
The constraints $\phi_{i,max} = 0.8$ is imposed by an upstream process and $\phi_{2,max} = \phi_{3,max} = 0.6$ is imposed by physical limits.

MPC Controller with Custom Input/Output Constraints

1 Open the Simulink model `mpc_blendingprocess`:



In the model, an MPC controller controls the blending process. The block labeled `Blending` incorporates the previously described model equations and includes unmeasured (step) disturbances in the feed compositions.



Signal u represents controller adjustments, i.e.,

$u = [\phi_T \; \phi_2 \; \phi_3]$.

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

Consider the following scenario:

- At $\tau = 0$, the process is operating at its nominal steady state.
- At $\tau = 0.5$, the demand decreases from $\phi = 1$ to $\phi = 0.9$.
- At $\tau = 1$, there is a large increase in the concentration of constituent 1 in feed 1 Y_{11} from 1.17 to 2.17.

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

2 Define a linear model.

```
% Create a linear approximation -- a state-space model based on the nominal
% operating point:
ni = 3;                     % number of feed streams
nc = 2;                     % number of components
Fin_nom = [1.6, 0.4, 0];    % Nominal flow rate for the ith feed stream
F_nom  = sum(Fin_nom);      % Nominal flow rate for the exit stream (demand)
cin_nom = [0.7 0.2 0.8      % Nominal composition for jth constituent in the ith feed flow
```

```
          0.3 0.8   0];
cout_nom = cin_nom*Fin_nom'/F_nom;  % Nominal product composition

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

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

3 Specify an MPC controller.

```
% Create the controller object with sampling period, prediction and control
% horizons:
Ts = 0.1;
p=10;
m=3;
MPCobj = mpc(Model, Ts, p, m);

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

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

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

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

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

4 Specify constraints.

Given the specified bounds on the feed 2 and 3 rates (= 0.6), it is possible that their sum could be as much as 1.2. Because the total feed rate is of order 0.9 to 1.0, the controller can request a physically impossible condition in which the sum of feeds 2 and 3 exceeds the total feed rate. This implies a negative feed 1 rate.

The constraint

$$0 \le \phi_1 = \phi_{T-\phi^2} - \phi_3 \le 0.8$$

prevents the controller from requesting an unrealistic ϕ_1 value.

To specify this constraint, enter:

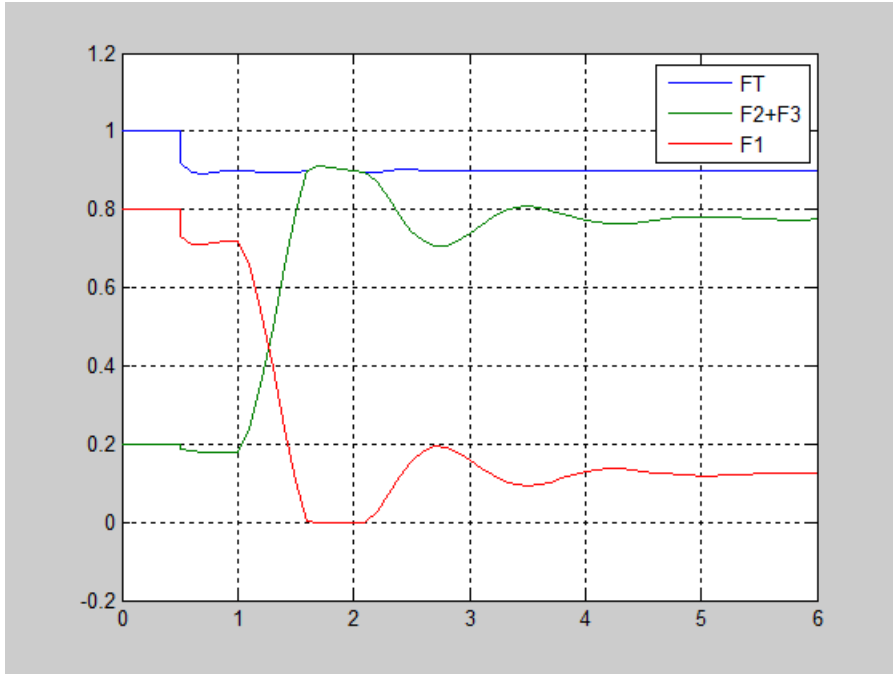
```
E = [-1 1 1; 1 -1 -1];

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

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

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

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

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

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

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

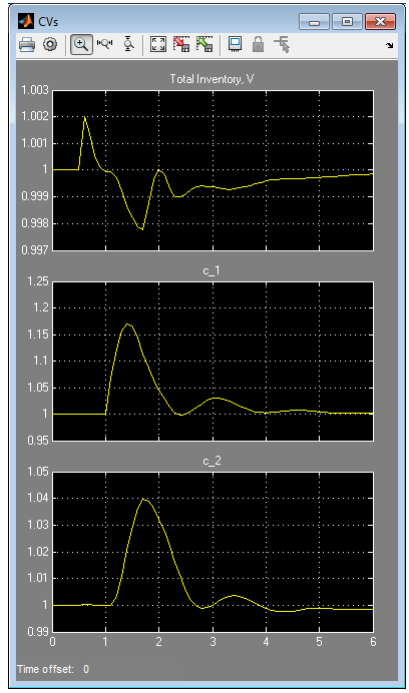
```
sim('mpc_blendingprocess')
figure
plot(MVs.time,[MVs.signals(1).values(:,2), ...
    (MVs.signals(2).values + MVs.signals(3).values), ...
    (MVs.signals(1).values(:,2)-MVs.signals(2).values-MVs.signals(3).values)])
grid
legend('FT','F2+F3','F1')
```

The plot shows the evolution of the total feed rate (blue curve) and the sum of feeds 2 and 3 (green curve). They coincide between $\tau = 1.7$ and $\tau = 2.2$.



If the custom input constraints had not been included, the controller would have requested a negative feed 1 rate during this period, as shown in by the red curve.

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



Related Examples

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

More About

"Constraints on Linear Combinations of Inputs and Outputs" on page 2-25

Providing LQR Performance Using Terminal Penalty

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

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

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

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

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

  You can shift the MPC prediction model's origin to eliminate this difference and achieve zero setpoints at nominal condition.
- The LQR cost function uses an infinite prediction horizon in which the manipulated variable changes at each sampling instant. In the standard MPC cost function, the horizon length is p, and the manipulated variable changes m times, where m is the control horizon.

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

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

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

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

1. Define a plant with one input and two outputs.

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

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

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

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

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

3 Design an MPC controller equivalent to the LQR controller.

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

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

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

b Create an MPC controller with equal prediction and control horizons.

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

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

```
for output(s) y1 and zero weight for output(s) y2 y3 y4
```

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

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

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

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

d Specify terminal weights.

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

```
U = struct('Weight', uwt);
Y = struct('Weight', [0 0 1 1]);
setterminal(MPCobj, Y, U);
```

The first two states receive zero weight at the terminal point, and the input weight is unchanged.

e Remove default state estimator.

The model states are measured directly, so the default MPC state estimator is unnecessary.

```
setoutdist(MPCobj,'model',tf(zeros(4,1)));
setEstimator(MPCobj,[],C);
```

The `setoutdist` command removes the output disturbances from the output channels, and the `setEstimator` command sets the controller state estimates equal to the measured output values.

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

a Compute closed-loop response with LQR controller.

```
clsys = feedback(Plant,K);
```

```
Tstop = 6;
x0 = [0.2; 0.2];
[yLQR tLQR] = initial(clsys,x0,Tstop);
```

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

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

-->The "Model.Noise" property of the "mpc" object is empty. Assuming white nois
```

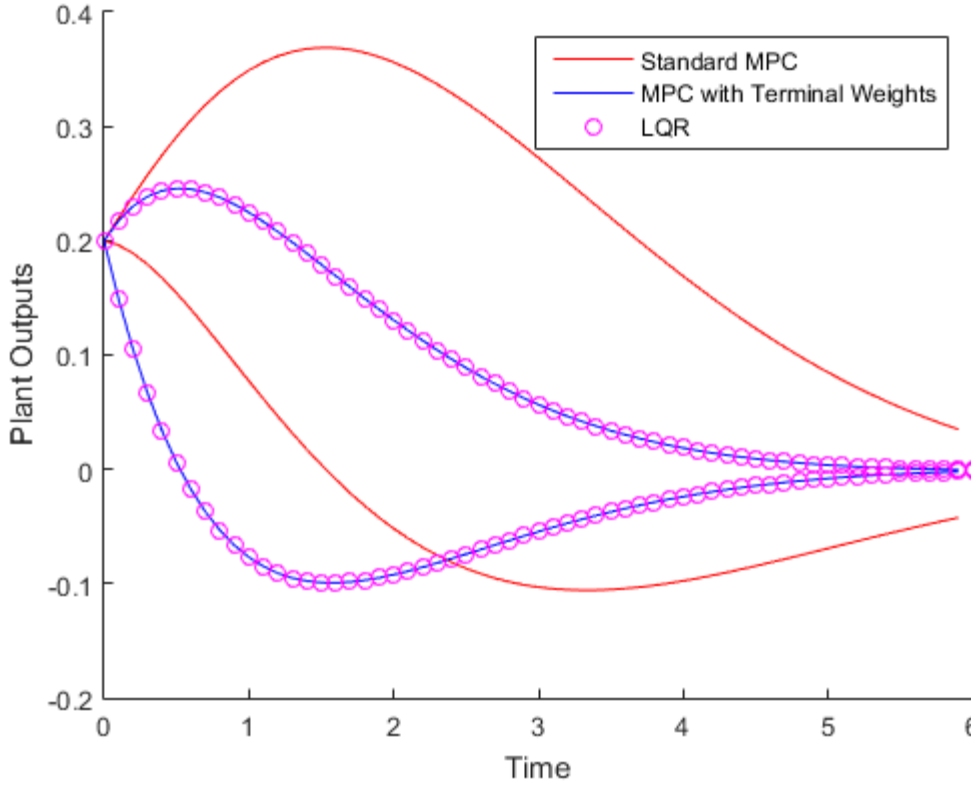
c Compute closed-loop response with standard MPC controller.

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

-->The "PredictionHorizon" property of "mpc" object is empty. Trying Prediction
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assumi
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. As
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming de
   for output(s) y1 and zero weight for output(s) y2
-->Integrated white noise added on measured output channel #1.
   Assuming unmeasured input disturbance #2 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white nois
```

d Compare the responses.

```
figure;
h1 = line(tSTD,ySTD,'color','r');
Annotation = get(h1,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');

h2 = line(t,y(:,1:2),'color','b');
Annotation = get(h2,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');
```

```
h3 = line(tLQR,yLQR,'color','m','marker','o','linestyle','none');
Annotation = get(h3,'Annotation');
set(get(Annotation{2},'LegendInformation'),'IconDisplayStyle','off');

xlabel('Time');
ylabel('Plant Outputs');
legend('Standard MPC','MPC with Terminal Weights','LQR','Location','NorthEast')
```



The plot shows that the MPC controller with the terminal weights provides faster settling to the origin than the standard MPC. The LQR controller and MPC with terminal weights provide identical control performance.

As reported by Scokaert and Rawlings [1], the computed `Cost` value is 2.23, identical to that provided by the LQR controller. The computed `CostSTD` value for the standard MPC is 4.82, more than double compared to `Cost`.

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

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

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

References

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

Related Examples

- "Designing Model Predictive Controller Equivalent to Infinite-Horizon LQR"

More About

- "Terminal Weights and Constraints"

Real-Time Control with OPC Toolbox

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

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

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

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

This example needs OPC Toolbox™.

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

```
The example needs OPC Toolbox(TM).
```

Establish a Connection to the OPC Server

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

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

Set up the Plant OPC I/O

In practice the plant would be a physical process, and the OPC tags which define its I/O would already have been created on the OPC server. However, since in this case

a simulation OPC server is being used, the plant behavior must be simulated. This is achieved by defining tags for the plant manipulated and measured variables and creating a callback (mpcopcPlantStep) to simulate plant response to changes in the manipulated variables. Two OPC groups are required, one to represent the two manipulated variables to be read by the plant simulator and another to write back the two measured plant outputs storing the results of the plant simulation.

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

Specify the MPC Controller Which Will Control the Simulated Plant

Create plant model.

```
plant_model = ss([-.2 -.1; 0 -.05],eye(2,2),eye(2,2),zeros(2,2));
disc_plant_model = c2d(plant_model,1);
% We assume no model mismatch, a control horizon 6 samples and
% prediction horizon 20 samples.
mpcobj = mpc(disc_plant_model,1,20,6);
mpcobj.weights.ManipulatedVariablesRate = [1 1];
% Build an internal MPC object structure so that the MPC object
% is not rebuilt each callback execution.
state = mpcstate(mpcobj);
y1 = mpcmove(mpcobj,state,[1;1]',[1 1]');
```

```
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
-->Integrated white noise added on measured output channel #1.
-->Integrated white noise added on measured output channel #2.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

Build the OPC I/O for the MPC Controller

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

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

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

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

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

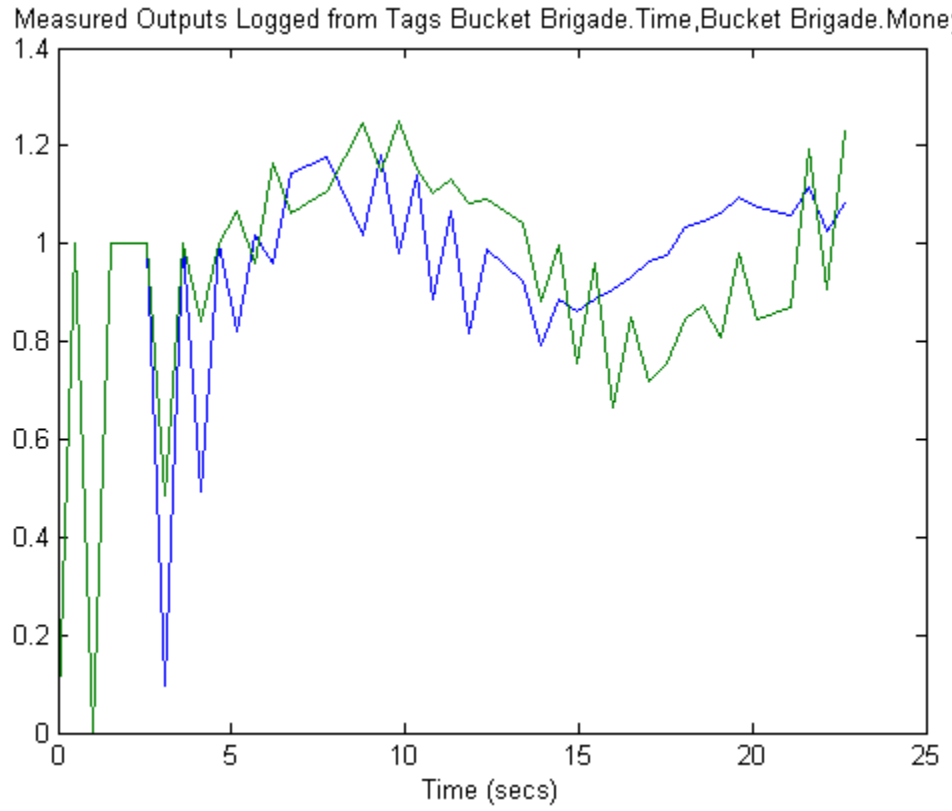
Log Data from the Plant Measured Outputs

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

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

Extract and Plot the Logged Data

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


Measured Outputs Logged from Tags Bucket Brigade.Time,Bucket Brigade.Money
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
Time (secs)

Simulation and Code Generation Using Simulink Coder

This example shows how to simulate and generate real-time code for an MPC Controller block with Simulink Coder. Code can be generated in both single and double precisions.

Required Products

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

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

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

Setup Environment

You must have write-permission to generate the relevant files and the executable. So, before starting simulation and code generation, change the current directory to a temporary directory.

```
cwd = pwd;
tmpdir = tempname;
mkdir(tmpdir);
cd(tmpdir);
```

Define Plant Model and MPC Controller

Define a SISO plant.

```
plant = ss(tf([3 1],[1 0.6 1]));
```

Define the MPC controller for the plant.

```
Ts = 0.1;   %Sampling time
p = 10;     %Prediction horizon
m = 2;      %Control horizon
Weights = struct('MV',0,'MVRate',0.01,'OV',1); % Weights
```

```
MV = struct('Min',-Inf,'Max',Inf,'RateMin',-100,'RateMax',100); % Input constraints
OV = struct('Min',-2,'Max',2); % Output constraints
mpcobj = mpc(plant,Ts,p,m,Weights,MV,OV);
```

Simulate and Generate Code in Double-Precision

By default, MPC Controller blocks use double-precision in simulation and code generation.

Simulate the model in Simulink.

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

The controller effort and the plant output are saved into base workspace as variables **u** and **y**, respectively.

Build the model with the `rtwbuild` command.

```
disp('Generating C code... Please wait until it finishes.');
set_param(mdl1,'RTWVerbose','off');
rtwbuild(mdl1);
```

On a Windows system, an executable file named "mpc_rtwdemo.exe" appears in the temporary directory after the build process finishes.

Run the executable.

```
if ispc
    disp('Running executable...');
    status = system(mdl1);
else
    disp('The example only runs the executable on Windows system.');
end
```

After the executable completes successfully (status=0), a data file named "mpc_rtwdemo.mat" appears in the temporary directory.

Compare the responses from the generated code (**rt_u** and **rt_y**) with the responses from the previous simulation in Simulink (**u** and **y**).

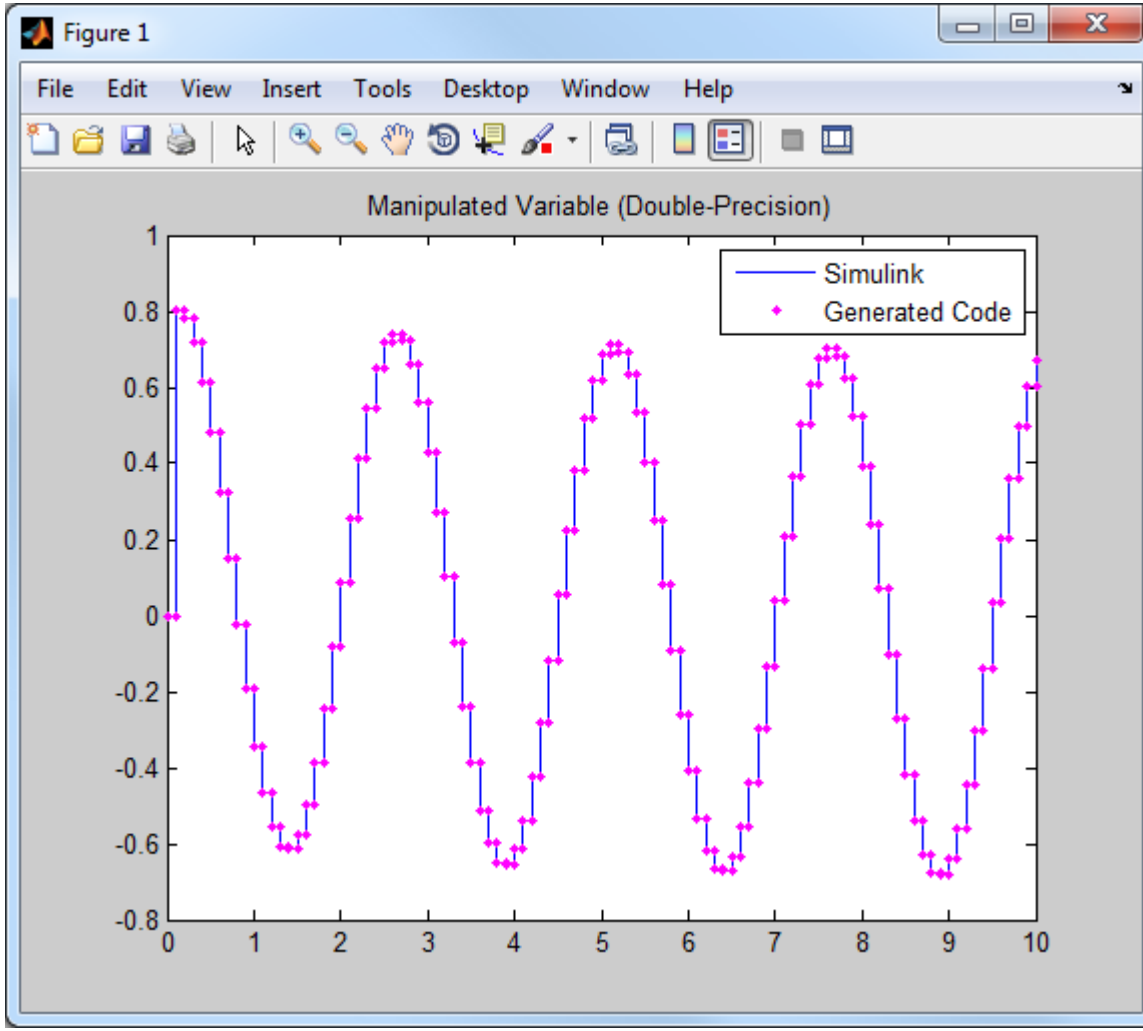
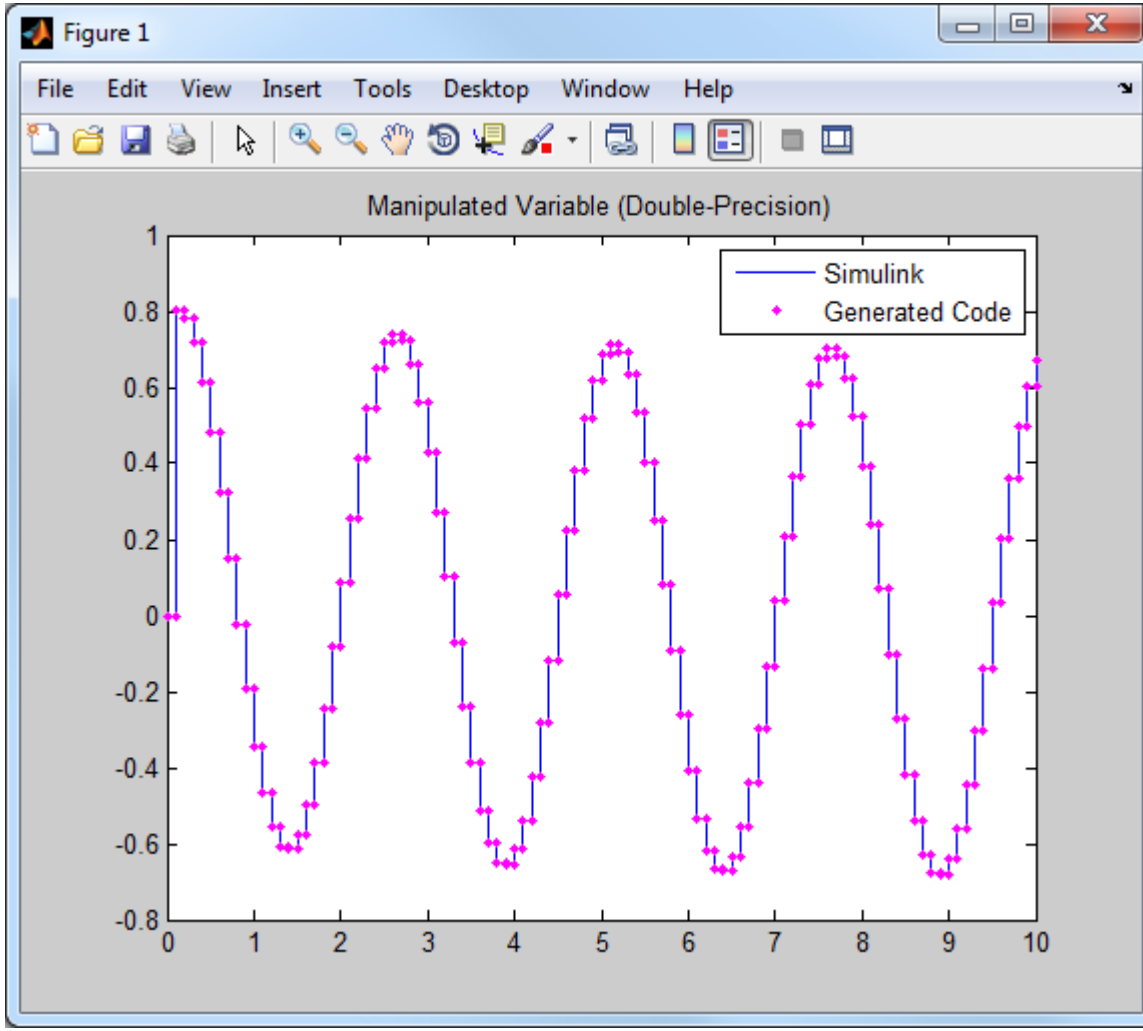


Figure 1
File Edit View Insert Tools Desktop Window Help
Manipulated Variable (Double-Precision)
Simulink
Generated Code
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0 1 2 3 4 5 6 7 8 9 10



They are numerically equal.

Simulate and Generate Code in Single-Precision

You can also configure the MPC block to use single-precision in simulation and code generation.

```
mdl2 = 'mpc_rtwdemo_single';
```

```
open_system(mdl2);
```

To do that, open the MPC block dialog and select "single" as the "output data type" at the bottom of the dialog.

```
open_system([mdl2 '/MPC Controller']);
```

Simulate the model in Simulink.

```
close_system([mdl2 '/MPC Controller']);
sim(mdl2);
```

The controller effort and the plant output are saved into base workspace as variables **u1** and **y1**, respectively.

Build the model with the `rtwbuild` command.

```
disp('Generating C code... Please wait until it finishes.');
set_param(mdl2,'RTWVerbose','off');
rtwbuild(mdl2);
```

On a Windows system, an executable file named "mpc_rtwdemo_single.exe" appears in the temporary directory after the build process finishes.

Run the executable.

```
if ispc
    disp('Running executable...');
    status = system(mdl2);
else
    disp('The example only runs the executable on Windows system.');
end
```

After the executable completes successfully (status=0), a data file named "mpc_rtwdemo_single.mat" appears in the temporary directory.

Compare the responses from the generated code (**rt_u1** and **rt_y1**) with the responses from the previous simulation in Simulink (**u1** and **y1**).

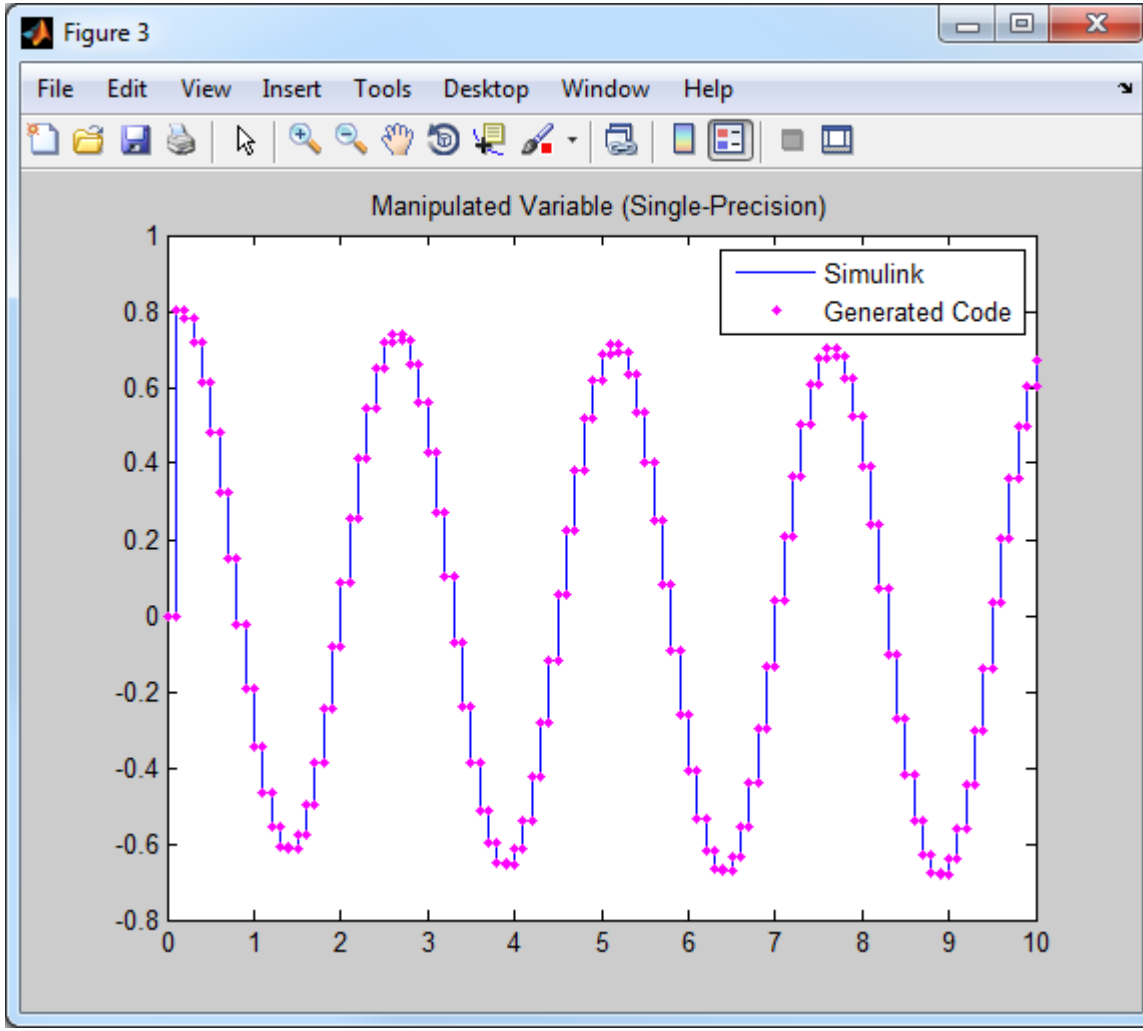
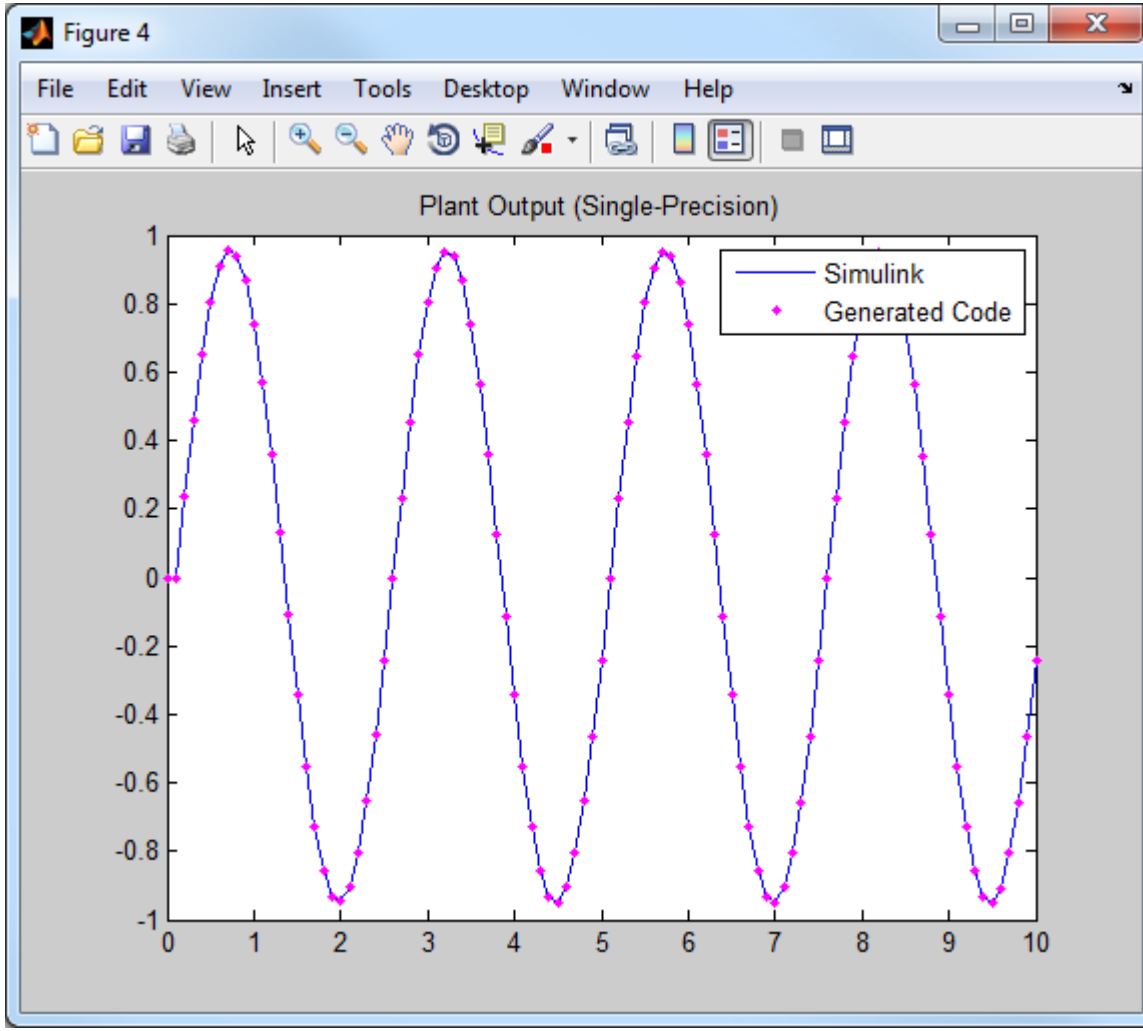


Figure 3
File Edit View Insert Tools Desktop Window Help
Manipulated Variable (Single-Precision)
Simulink
Generated Code
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0 1 2 3 4 5 6 7 8 9 10



They are numerically equal.

Close the Simulink model.

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

cd(cwd)
```

Simulation and Structured Text Generation Using PLC Coder

This example shows how to simulate and generate Structured Text for an MPC Controller block using PLC Coder software. The generated code uses single-precision.

Required Products

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

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

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

Setup Environment

You must have write-permission to generate the relevant files and the executable. So, before starting simulation and code generation, change the current directory to a temporary directory.

```
cwd = pwd;
tmpdir = tempname;
mkdir(tmpdir);
cd(tmpdir);
```

Define Plant Model and MPC Controller

Define a SISO plant.

```
plant = ss(tf([3 1],[1 0.6 1]));
```

Define the MPC controller for the plant.

```
Ts = 0.1;   %Sampling time
p = 10;     %Prediction horizon
m = 2;      %Control horizon
Weights = struct('MV',0,'MVRate',0.01,'OV',1); % Weights
```

```
MV = struct('Min',-Inf,'Max',Inf,'RateMin',-100,'RateMax',100); % Input constraints
OV = struct('Min',-2,'Max',2); % Output constraints
mpcobj = mpc(plant,Ts,p,m,Weights,MV,OV);
```

Simulate and Generate Structured Text

Open the Simulink model.

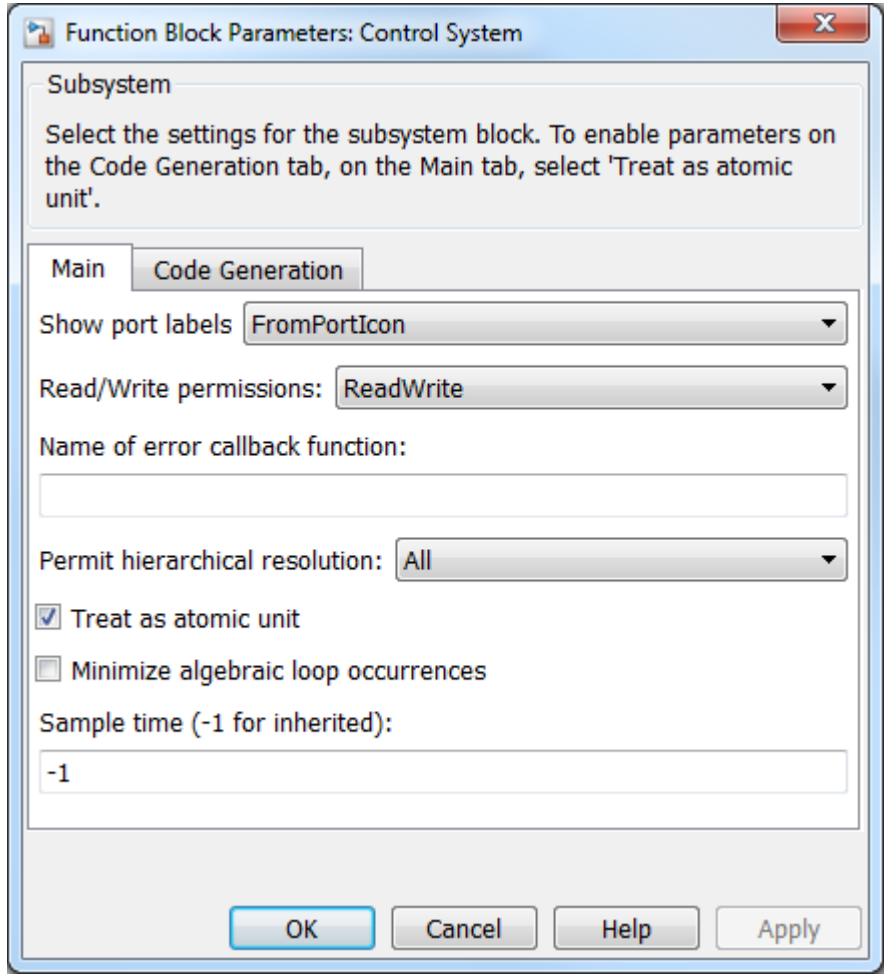
```
mdl = 'mpc_plcdemo';
open_system(mdl);
```

To generate structured text for the MPC Controller block, complete the following two steps:

- Configure the MPC block to use single precision. Select "single" in the "Output data type" combo box in the MPC block dialog.

```
open_system([mdl '/Control System/MPC Controller']);
```

- Put MPC block inside a subsystem block and treat the subsystem block as an atomic unit. Select the "Treat as atomic unit" checkbox in the subsystem block dialog.



Simulate the model in Simulink.

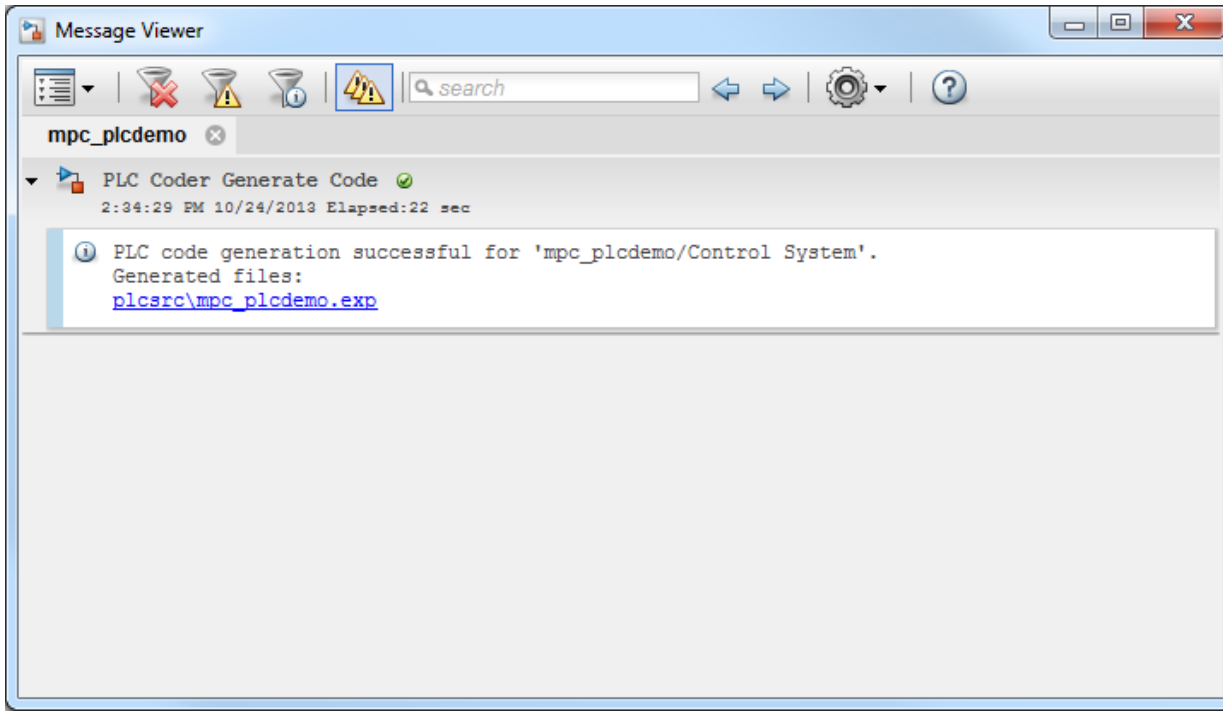
```
close_system([mdl '/Control System/MPC Controller']);
open_system([mdl '/Outputs//References']);
open_system([mdl '/Inputs']);
sim(mdl);
```

To generate code with the PLC Coder, use the `plcgeneratecode` command.

```
disp('Generating PLC structure text... Please wait until it finishes.');
```

```
plcgeneratecode([mdl '/Control System']);
```

The Message Viewer dialog box shows that PLC code generation was successful.



Close the Simulink model.

```
bdclose(mdl);
```

```
cd(cwd)
```

Setting Targets for Manipulated Variables

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

Define Plant Model

The linear plant model has two inputs and two outputs.

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

Design MPC Controller

Create MPC controller.

```
Ts = 0.4;                       % Sampling time
mpcobj = mpc(plant,Ts,20,5);
```

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

Specify weights.

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

Define input specifications.

```
mpcobj.MV = struct('RateMin',{-0.5;-0.5},'RateMax',{0.5;0.5});
```

Specify target set-point u=2 for the first manipulated variable.

```
mpcobj.MV(1).Target=2;
```

Simulation Using Simulink®

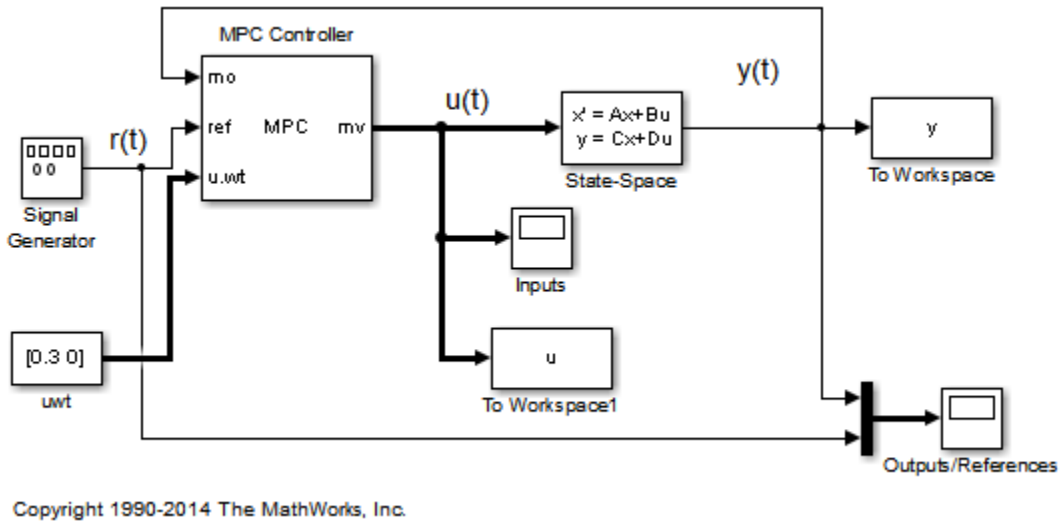
To run this example, Simulink® is required.

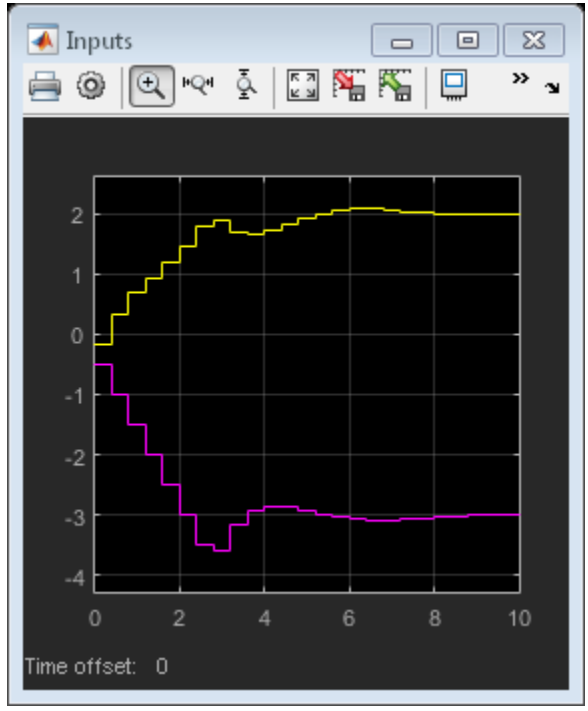
```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
```

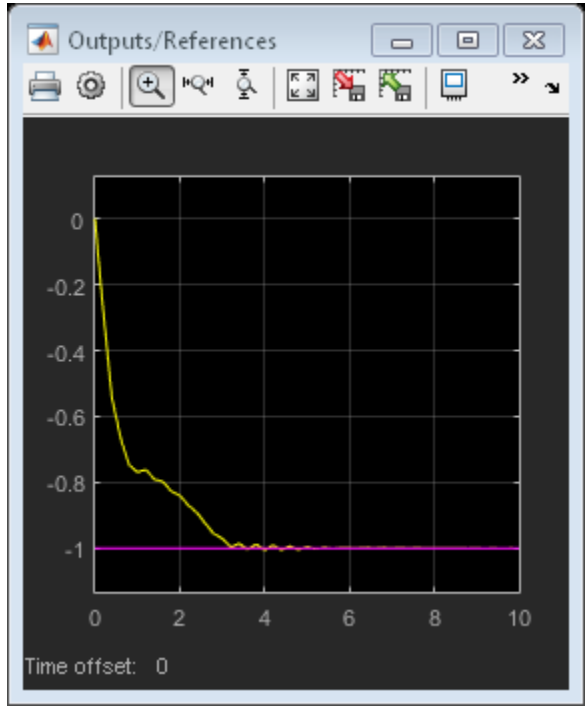
Simulate.

```
mdl = 'mpc_utarget';
open_system(mdl)      % Open Simulink(R) Model
sim(mdl);             % Start Simulation
```

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




Inputs
2
1
0
-1
-2
-3
-4
0
2
4
6
8
10
Time offset: 0




```
bdclose(mdl)
```

Specifying Alternative Cost Function with Off-Diagonal Weight Matrices

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

Define Plant Model and MPC Controller

The linear plant model has two inputs and two outputs.

```
plant = ss(tf({1,1;1,2},{[1 .5 1],[.7 .5 1];[1 .4 2],[1 2]}));
[A,B,C,D] = ssdata(plant);
Ts = 0.1;                % sampling time
plant = c2d(plant,Ts);  % convert to discrete time
```

Create MPC controller.

```
p=20;       % prediction horizon
m=2;        % control horizon
mpcobj = mpc(plant,Ts,p,m);
```

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

Define constraints on the manipulated variable.

```
mpcobj.MV = struct('Min',{-3;-2},'Max',{3;2},'RateMin',{-100;-100},'RateMax',{100;100});
```

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

```
OW = [1 -1]'*[1 -1];
% Non-diagonal output weight, corresponding to ((y1-r1)-(y2-r2))^2
mpcobj.Weights.OutputVariables = {OW};
% Non-diagonal input weight, corresponding to (u1-u2)^2
mpcobj.Weights.ManipulatedVariables = {0.5*OW};
```

Simulate Using SIM Command

Specify simulation options.

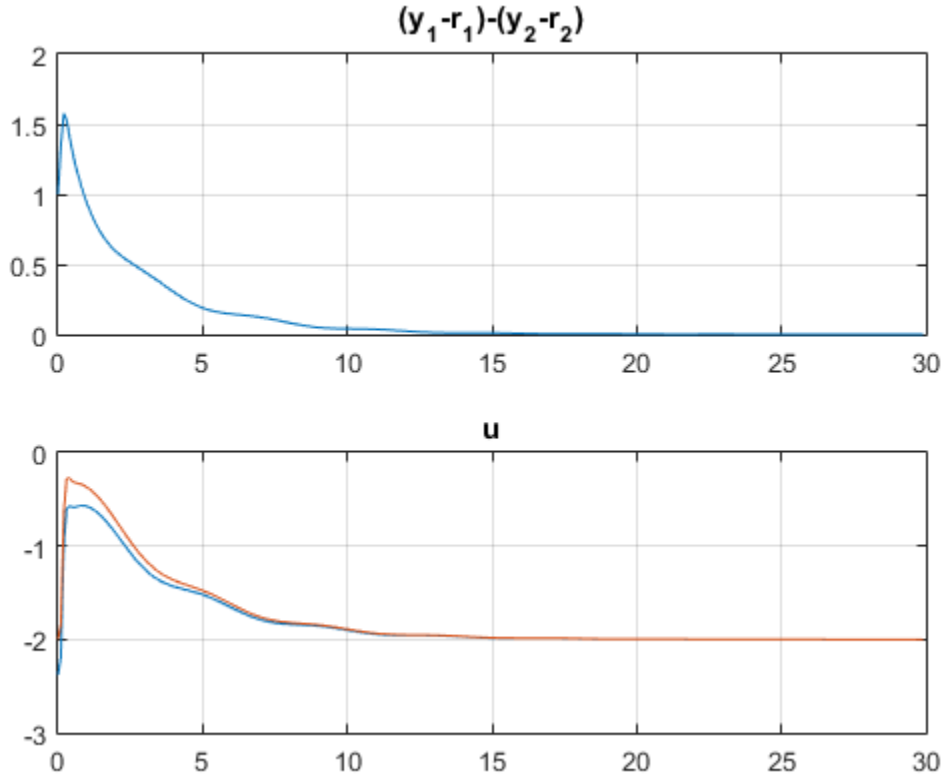
```
Tstop = 30;               % simulation time
Tf = round(Tstop/Ts);     % number of simulation steps
```

```
r = ones(Tf,1)*[1 2];      % reference trajectory
```

Run the closed-loop simulation and plot results.

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

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



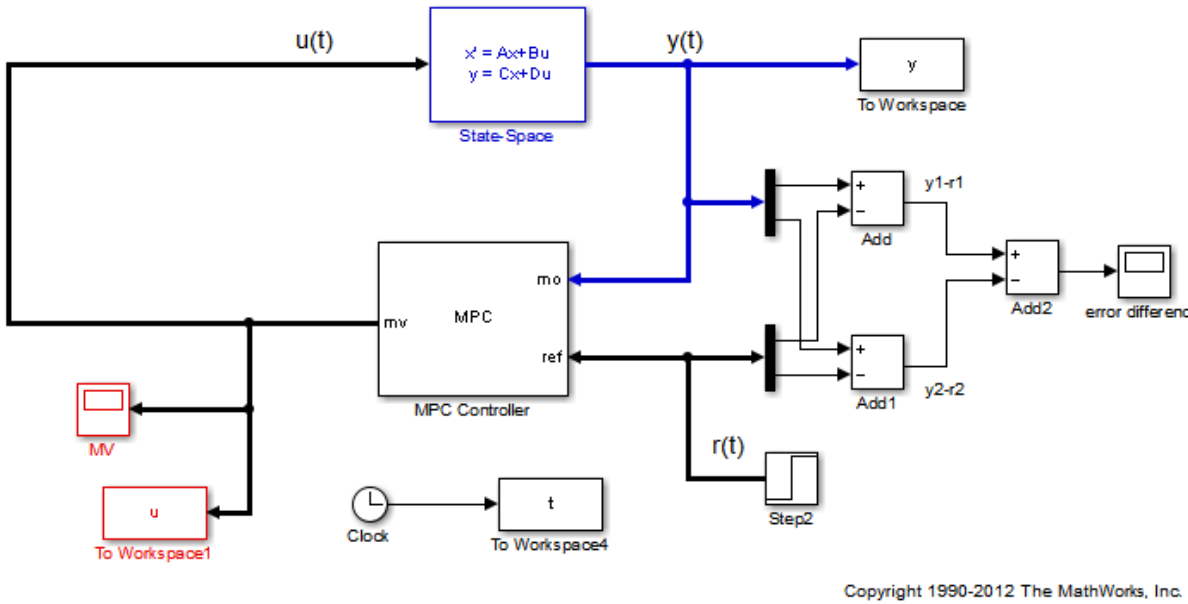
Simulate Using Simulink®

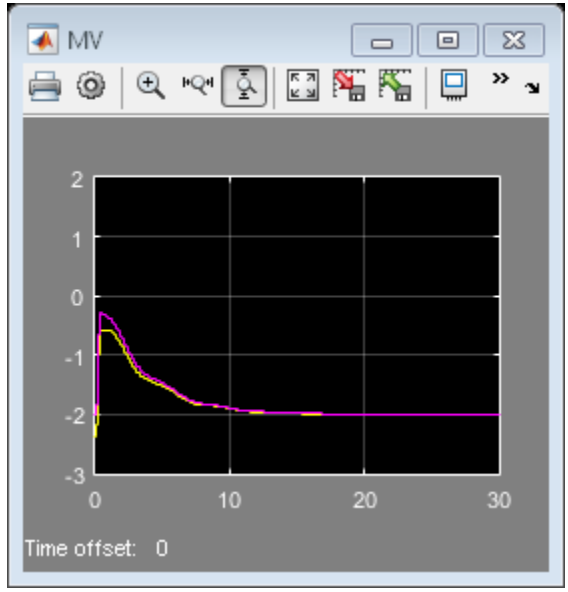
To run this example, Simulink® is required.

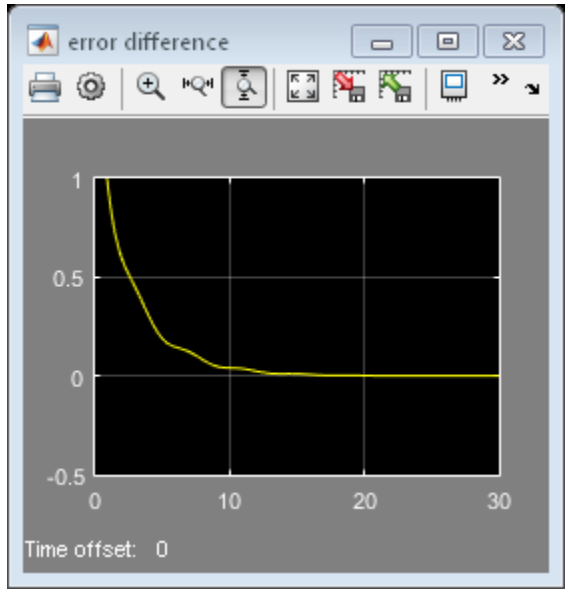
```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this part of the example.')
    return
end
```

Now simulate closed-loop MPC in Simulink®.

```
mdl = 'mpc_weightsdemo';
open_system(mdl);
sim(mdl)
```




MV
2
1
0
-1
-2
-3
0
10
20
30
Time offset: 0


error difference
1
0.5
0
-0.5
0
10
20
30
Time offset: 0

```
bdclose(mdl);
```

Review Model Predictive Controller for Stability and Robustness Issues

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

The Fuel Gas Blending Process

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

Inputs:

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

Outputs:

```
1. High Heating Value (HHV)
2. Wobbe Index (WI)
3. Flame Speed Index (FSI)
4. Header Pressure (P)
```

The fuel gas blending process was studied by Muller et al.: "Modeling, validation, and control of an industrial fuel gas blending system", C.J. Muller, I.K. Craig, N.L. Ricker, J. of Process Control, in press, 2011.

Linear Plant Model

Use the following linear plant model as the prediction model for the controller. This state-space model, applicable at a typical steady-state operating point, uses the time unit of hours.

```
a = diag([-28.6120, -28.6822, -28.5134,  -0.0281, -23.2191, -23.4266, ...
```

```
          -22.9377, - 0.0101, -26.4877, -26.7950, -27.2210,  -0.0083, ...
          -23.0890, -23.0062, -22.9349,  -0.0115, -25.8581, -25.6939, ...
          -27.0793,  -0.0117, -22.8975, -22.8233, -21.1142,  -0.0065]);
b = zeros(24,6);
b( 1: 4,1) = [4, 4, 8, 32]';
b( 5: 8,2) = [2, 2, 4, 32]';
b( 9:12,3) = [2, 2, 4, 32]';
b(13:16,4) = [4, 4, 8, 32]';
b(17:20,5) = [2, 2, 4, 32]';
b(21:24,6) = [1, 2, 1, 32]';
c = [diag([ 6.1510,  7.6785, -5.9312, 34.2689]), ...
     diag([-2.2158, -3.1204,  2.6220, 35.3561]), ...
     diag([-2.5223,  1.1480,  7.8136, 35.0376]), ...
     diag([-3.3187, -7.6067, -6.2755, 34.8720]), ...
     diag([-1.6583, -2.0249,  2.5584, 34.7881]), ...
     diag([-1.6807, -1.2217,  1.0492, 35.0297])];
d = zeros(4,6);
Plant = ss(a, b, c, d);
```

By default, all the plant inputs are manipulated variables.

```
Plant.InputName = {'NG', 'RG', 'H2', 'N2', 'T1', 'T2'};
```

By default, all the plant outputs are measured outputs.

```
Plant.OutputName = {'HHV', 'WI', 'FSI', 'P'};
```

Transport delay is added to plant outputs to reflect the delay in the sensors.

```
Plant.OutputDelay = [0.00556  0.0167  0.00556  0];
```

Initial Controller Design

Construct an initial model predictive controller based on design requirements.

Specify sampling time, horizons and steady-state values.

The sampling time is that of the sensors (20 seconds). The prediction horizon is approximately equal to the plant settling time (39 intervals). The control horizon uses four blocked moves that have lengths of 2, 6, 12 and 19 intervals respectively. The nominal operating conditions are non-zero. The output measurement noise is white noise with magnitude of 0.001.

```
MPC_verbosity = mpcverbosity('off'); % Disable MPC message displaying at command line
```

```
Ts = 20/3600;   % Time units are hours.
Obj = mpc(Plant, Ts, 39, [2, 6, 12, 19]);
Obj.Model.Noise = ss(0.001*eye(4));
Obj.Model.Nominal.Y = [16.5, 25, 43.8, 2100];
Obj.Model.Nominal.U = [1.4170, 0, 2, 0, 0, 26.5829];
```

Specify lower and upper bounds on manipulated variables.

Since all the manipulated variables are flow rates of gas streams, their lower bounds are zero. All the MV constraints are hard (MinECR and MaxECR = 0) by default.

```
MVmin = zeros(1,6);
MVmax = [15, 20, 5, 5, 30, 30];
for i = 1:6
    Obj.MV(i).Min = MVmin(i);
    Obj.MV(i).Max = MVmax(i);
end
```

Specify lower and upper bounds on manipulated variable increments.

The bounds are set large enough to allow full range of movement in one interval. All the MV rate constraints are hard (RateMinECR and RateMaxECR = 0) by default.

```
for i = 1:6
    Obj.MV(i).RateMin = -MVmax(i);
    Obj.MV(i).RateMax =  MVmax(i);
end
```

Specify lower and upper bounds on plant outputs.

All the OV constraints are soft (MinECR and MaxECR = 0) by default.

```
OVmin = [16.5, 25, 39, 2000];
OVmax = [18.0, 27, 46, 2200];
for i = 1:4
    Obj.OV(i).Min = OVmin(i);
    Obj.OV(i).Max = OVmax(i);
end
```

Specify weights on manipulated variables.

MV weights are specified based on the known costs of each feed stream. This tells MPC controller how to move the six manipulated variables in order to minimize the cost of the

blended fuel gas. The weights are normalized so the maximum weight is approximately 1.0.

```
Obj.Weights.MV = [54.9, 20.5, 0, 5.73, 0, 0]/55;
```

Specify weights on manipulated variable increments.

They are small relative to the maximum MV weight so the MVs are free to vary.

```
Obj.Weights.MVrate = 0.1*ones(1,6);
```

Specify weights on plant outputs.

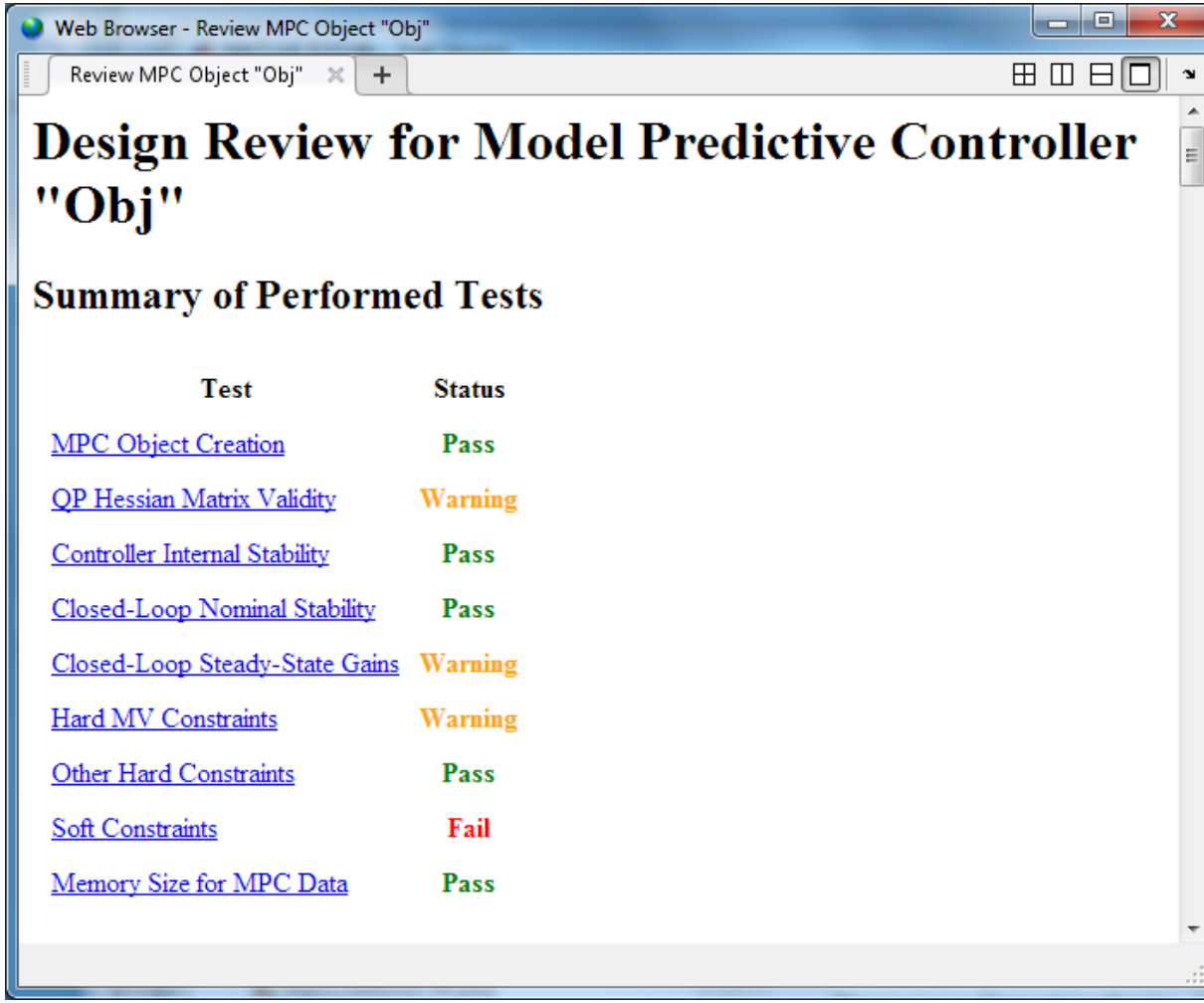
The OV weights penalize deviations from specified setpoints and would normally be "large" relative to the other weights. Let us first consider the default values, which equal the maximum MV weight specified above.

```
Obj.Weights.OV = [1, 1, 1, 1];
```

Using the `review` Command to Improve the Initial Design

Review the initial controller design.

```
review(Obj)
```



The summary table shown above lists three warnings and one error. Let's consider these in turn. Click **QP Hessian Matrix Validity** and scroll down to display the warning. It indicates that the plant signal magnitudes differ significantly. Specifically, the pressure response is much larger than the others.

Scale Factors

Scaling converts the relationship between output variables and manipulated variables to dimensionless form. Scale factor specifications can improve QP numerical accuracy. They also make it easier to specify tuning weight magnitudes.

In order for the outputs to be controllable, each must respond to at least one manipulated variable within the prediction horizon. If the plant is well scaled, the maximum absolute value of such responses should be of order unity.

Outputs whose maximum absolute scaled responses are outside the range [0.1,10] appear below. The table shows the maximum absolute response of each such OV with respect to each MV.

	NG	RG	H2	N2	T1	T2
P	236.876	244.868	242.709	241.478	240.892	242.702

Warning: at least one output variable response indicates poor scaling. Consider adjusting MV and OV ScaleFactors.

Examination of the specified OV bounds shows that the spans are quite different, and the pressure span is two orders of magnitude larger than the others. It is good practice to specify MPC scale factors to account for the expected differences in signal magnitudes. We are already weighting MVs based on relative cost, so we focus on the OVs only.

Calculate OV spans

```
OVspan = OVmax - OVmin;
%
% Use these as the specified scale factors
for i = 1:4
    Obj.OV(i).ScaleFactor = OVspan(i);
end
% Use review to verify that the scale factor warning has disappeared.
review(Obj);
%
% <<reviewDemo03.png>>
```

The next warning indicates that the controller does not drive the OVs to their targets at steady state. Click **Closed-Loop Steady-State Gains** to see a list of the non-zero gains.

Closed-Loop Steady-State Gains

`cloffset` is used to determine whether the controller forces all controlled output variables to their targets at steady state, in the absence of constraints.

The command calculates the impact of a sustained disturbance on each measured output variable (OV) in terms of an input/output gain. If a gain is zero, the controller eliminates steady-state tracking error for that disturbance-to-output mapping.

The gains with magnitudes exceeding 1e-05 are as follows:

Disturbed OV	Affected OV	Gain
HHV	HHV	0.0860281
WI	HHV	-0.0344992
FSI	HHV	0.0665757
HHV	WI	-0.036145
WI	WI	0.014495
FSI	WI	-0.027972
HHV	FSI	0.279361
WI	FSI	-0.11203
FSI	FSI	0.216193
HHV	P	0.0468767
WI	P	-0.0187986
FSI	P	0.036277

Warning: your design allows non-zero steady-state tracking errors in at least one controlled output. If this was not your intent, possible causes are as follows:

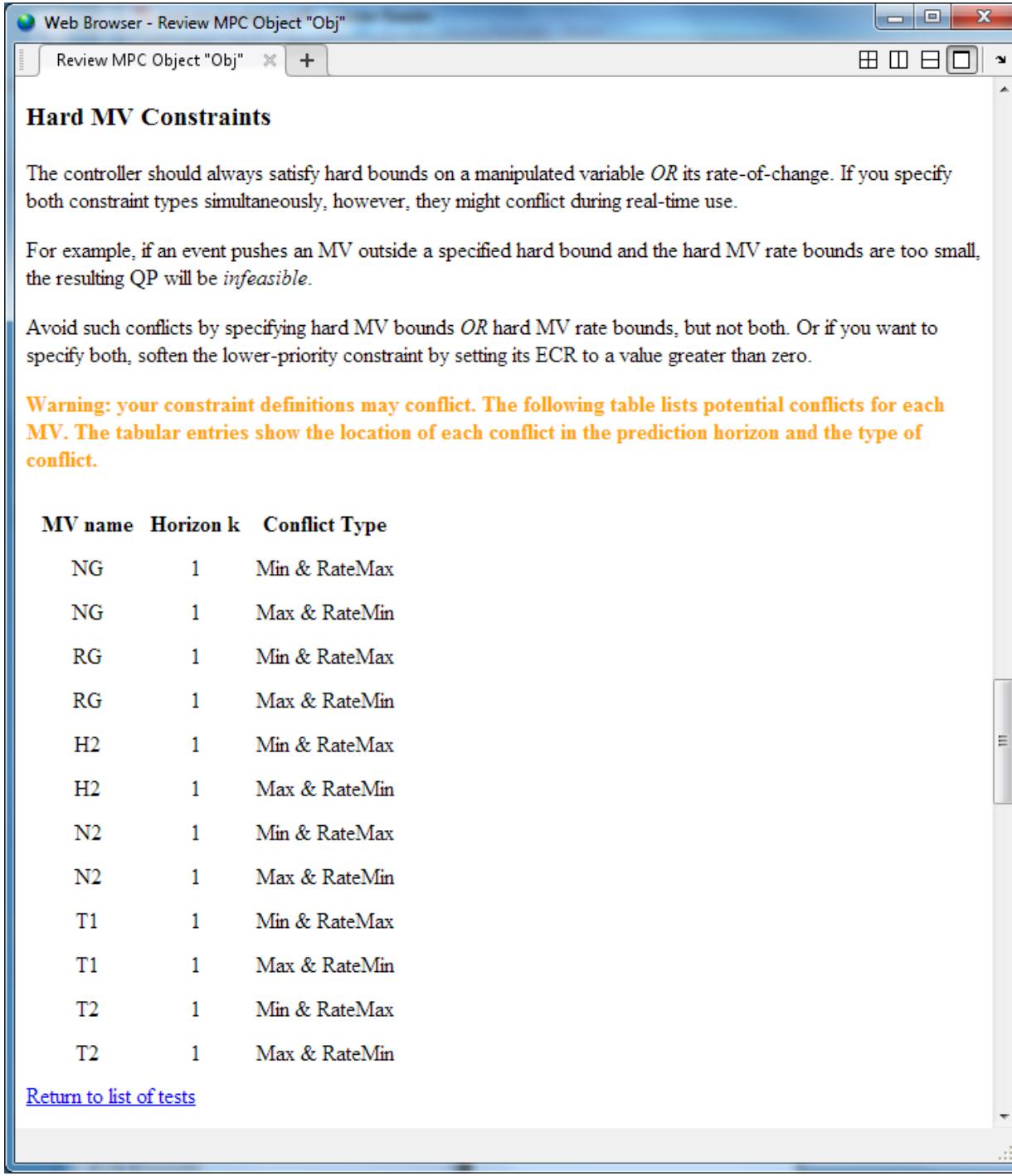
- Zero penalty weight on a plant output. Check the Weights.OV property.
- Non-zero penalty weight on a manipulated variable. Check the Weights.MV property.
- State estimator that does not include integration of output tracking error. The default estimator includes integration. If you have modified or replaced it, review your estimator design.

The first entry in the list shows that adding a sustained disturbance of unit magnitude to the HHV output would cause the HHV to deviate 0.0860 units from its steady-state target, assuming no constraints are active. The second entry shows that a unit disturbance in WI would cause a steady-state deviation ("offset") of -0.0345 in HHV, etc.

Since there are six MVs and only four OVs, excess degrees of freedom are available and you might be surprised to see non-zero steady-state offsets. The non-zero MV weights we have specified in order to drive the plant toward the most economical operating condition are causing this.

Non-zero steady-state offsets are often undesirable but are acceptable in this application because: # The primary objective is to minimize the blend cost. The gas quality (HHV, etc.) can vary freely within the specified OV limits. # The small offset gain magnitudes indicate that the impact of disturbances would be small. # The OV limits are soft constraints. Small, short-term violations are acceptable.

View the second warning by clicking **Hard MV Constraints**, which indicates a potential hard-constraint conflict.



Hard MV Constraints

The controller should always satisfy hard bounds on a manipulated variable *OR* its rate-of-change. If you specify both constraint types simultaneously, however, they might conflict during real-time use.

For example, if an event pushes an MV outside a specified hard bound and the hard MV rate bounds are too small, the resulting QP will be *infeasible*.

Avoid such conflicts by specifying hard MV bounds *OR* hard MV rate bounds, but not both. Or if you want to specify both, soften the lower-priority constraint by setting its ECR to a value greater than zero.

Warning: your constraint definitions may conflict. The following table lists potential conflicts for each MV. The tabular entries show the location of each conflict in the prediction horizon and the type of conflict.

MV name	Horizon k	Conflict Type
NG	1	Min & RateMax
NG	1	Max & RateMin
RG	1	Min & RateMax
RG	1	Max & RateMin
H2	1	Min & RateMax
H2	1	Max & RateMin
N2	1	Min & RateMax
N2	1	Max & RateMin
T1	1	Min & RateMax
T1	1	Max & RateMin
T2	1	Min & RateMax
T2	1	Max & RateMin

Return to list of tests

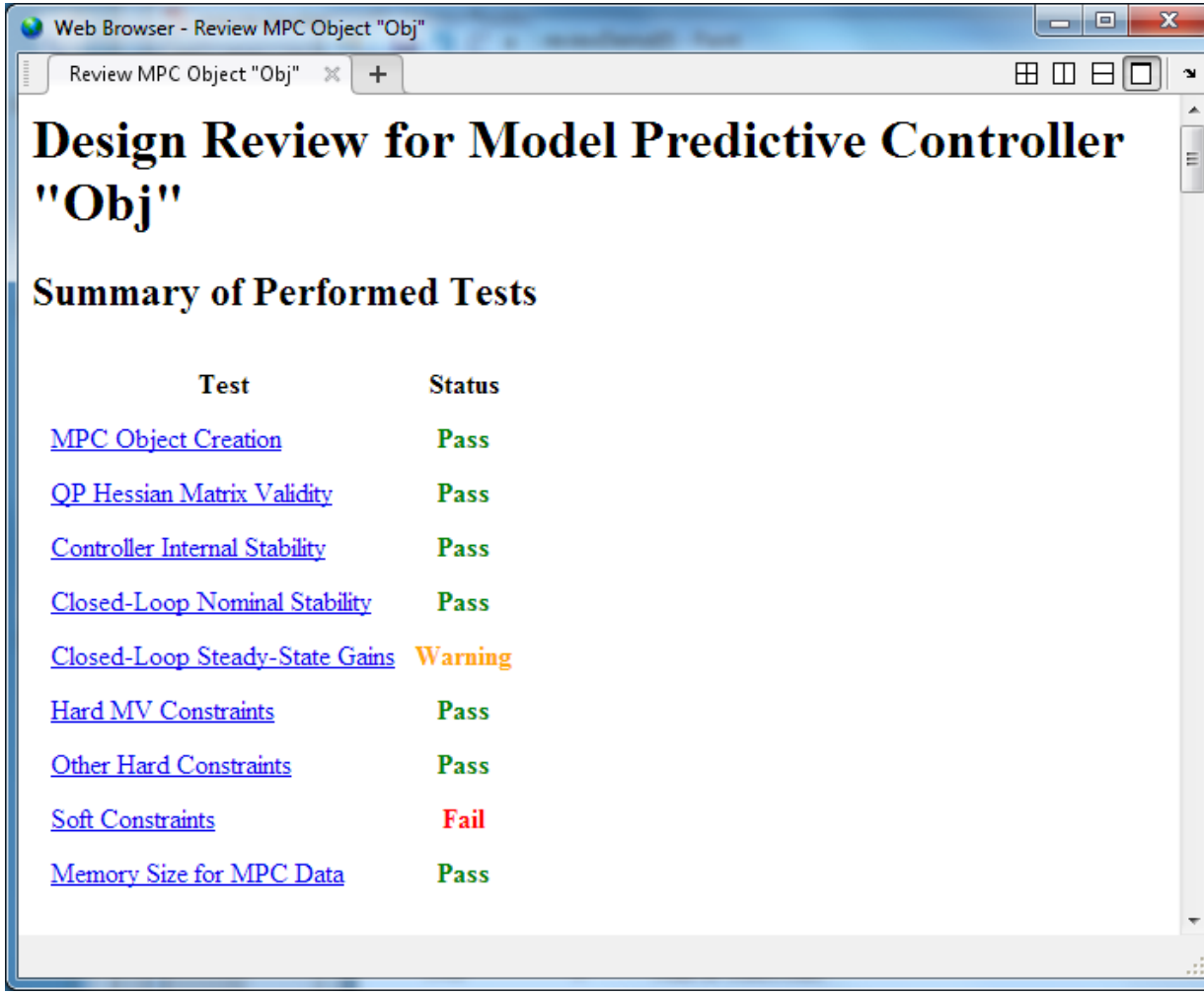
If an external event causes the NG to go far below its specified minimum, the constraint on its rate of increase might make it impossible to return the NG within bounds in one interval. In other words, when you specify both MV.Min and MV.RateMax, the controller would not be able to find an optimal solution if the most recent MV value is less than (MV.Min - MV.RateMax). Similarly, there is a potential conflict when you specify both MV.Max and MV.RateMin.

An MV constraint conflict would be extremely unlikely in the gas blending application, but it's good practice to eliminate the possibility by softening one of the two constraints. Since the MV minimum and maximum values are physical limits and the increment bounds are not, we soften the increment bounds as follows:

```
for i = 1:6
    Obj.MV(i).RateMinECR = 0.1;
    Obj.MV(i).RateMaxECR = 0.1;
end
```

Review the new controller design.

```
review(Obj)
```



The MV constraint conflict warning has disappeared. Now click **Soft Constraints** to view the error message.

Impact of delays

Delays can make it impossible to satisfy output constraints. The presence of unattainable constraints usually degrades performance. Let j be the location (within the prediction horizon) of the first finite constraint value (Min or Max) for OV(i). If all delays for OV(i) exceed j, the constraint is unattainable.

The following table lists each output constraint that is impossible to satisfy. The first column is the location (within the prediction horizon) of the first finite constraint value. The second column is the minimum delay for that output variable.

Constraint	Begins	Delay
WI.Min	1	3
WI.Max	1	3

Error: at least one output variable constraint is impossible to satisfy.

We see that the delay in the WI output makes it impossible to satisfy bounds on that variable until at least three control intervals have elapsed. The WI bounds are soft but it is poor practice to include unattainable constraints in a design. We therefore modify the WI bound specifications such that it is unconstained until the 4th prediction horizon step.

```
Obj.OV(2).Min = [-Inf(1,3), OVmin(2)];
Obj.OV(2).Max = [ Inf(1,3), OVmax(2)];

% Ee-issuing the review command to verifies that this eliminates the
% error message (see the next step).
```

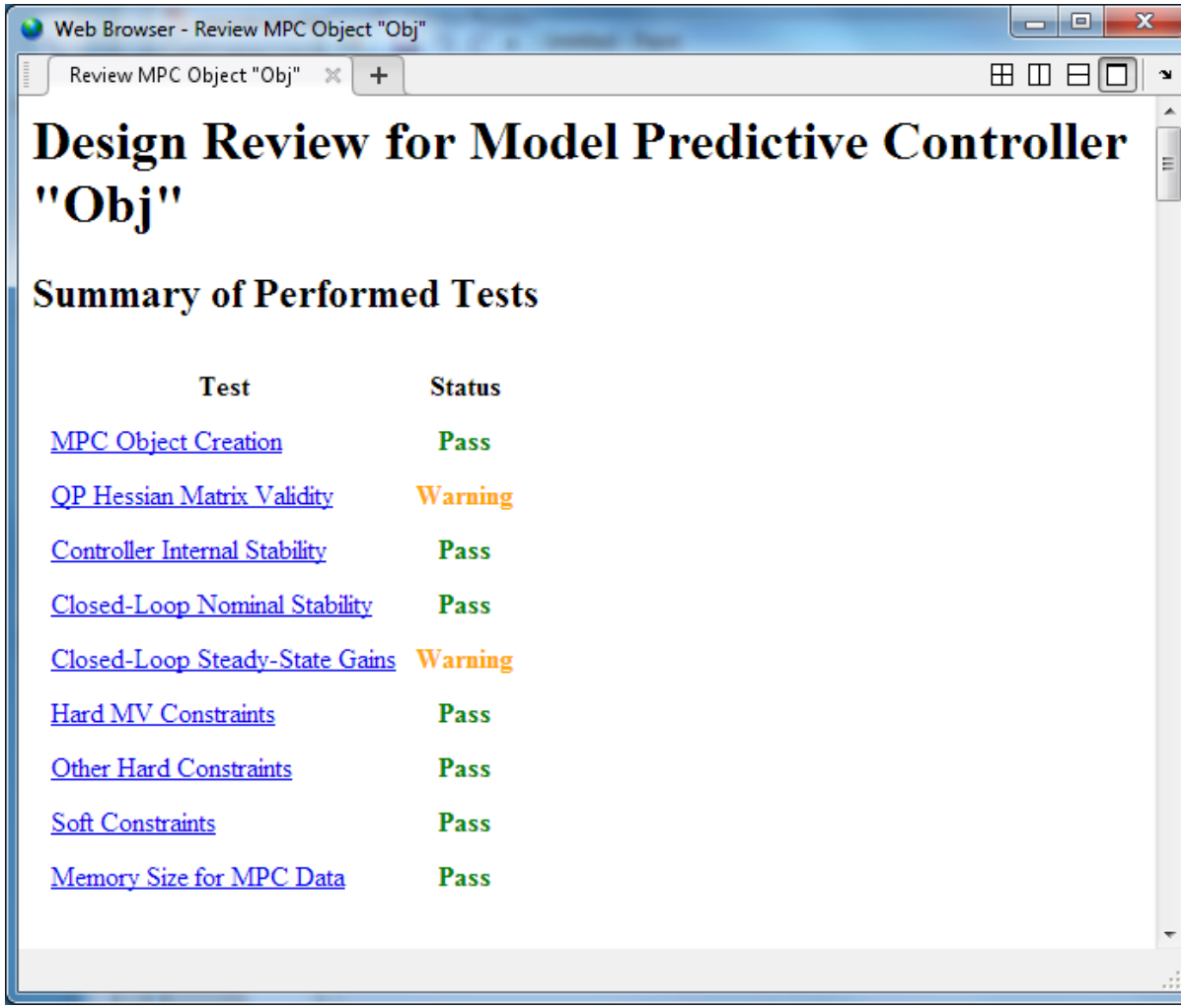
Diagnosing the Impact of Zero Output Weights

Given that the design requirements allow the OVs to vary freely within their limits, consider zeroing their penalty weights:

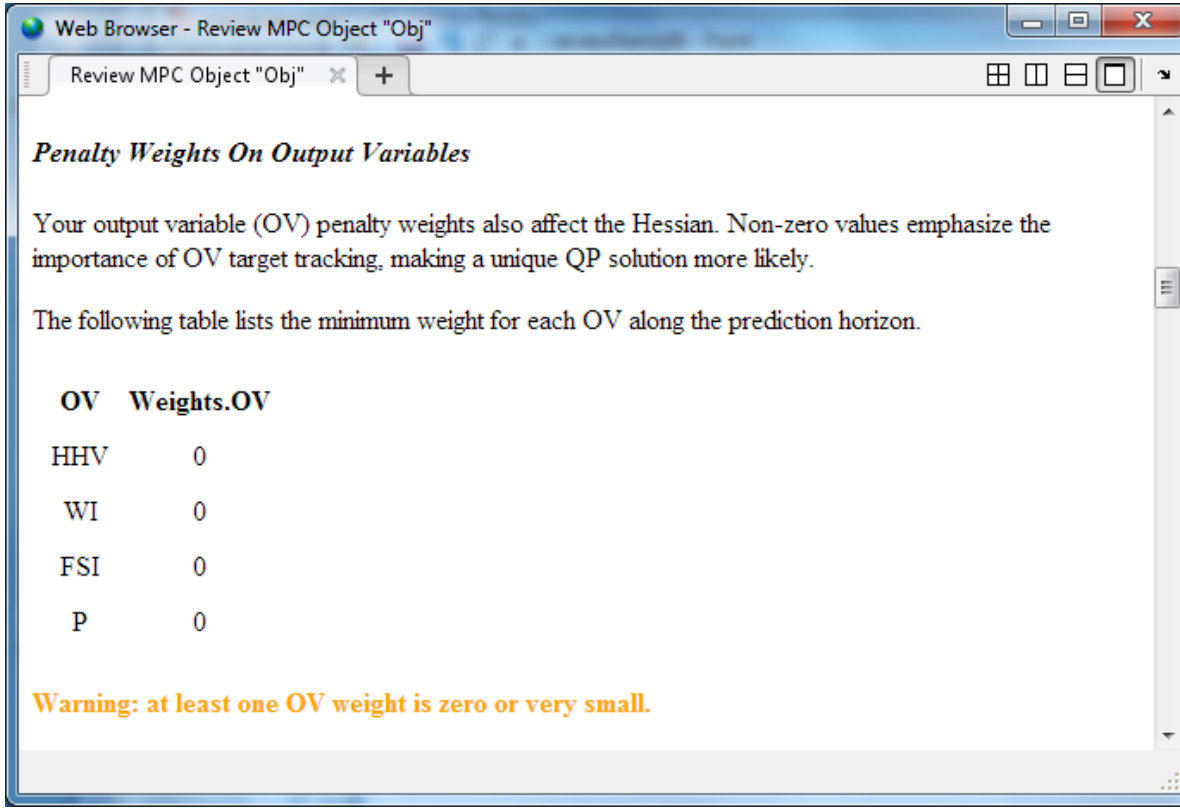
```
Obj.Weights.OV = zeros(1,4);
```

Review the impact of this design change.

```
review(Obj)
```



A new warning regarding QP Hessian Matrix Validity has appeared. Click **QP Hessian Matrix Validity** warning to see the details.



The review has flagged the zero weights on all four output variables. Since the zero weights are consistent with the design requirement and the other Hessian tests indicate that the quadratic programming problem has a unique solution, this warning can be ignored.

Click **Closed-Loop Steady-State Gains** to see the second warning. It shows another consequence of setting the four OV weights to zero. When an OV is not penalized by a weight, any output disturbance added to it will be ignored, passing through with no attenuation.

Closed-Loop Steady-State Gains

`cloffset` is used to determine whether the controller forces all controlled output variables to their targets at steady state, in the absence of constraints.

The command calculates the impact of a sustained disturbance on each measured output variable (OV) in terms of an input/output gain. If a gain is zero, the controller eliminates steady-state tracking error for that disturbance-to-output mapping.

The gains with magnitudes exceeding 1e-05 are as follows:

Disturbed OV	Affected OV	Gain
HHV	HHV	1
WI	WI	1
FSI	FSI	1
P	P	1

Since it is a design requirement, non-zero steady-state offsets are acceptable provided that MPC is able to hold all the OVs within their specified bounds. It is therefore a good idea to examine how easily the soft OV constraints can be violated when disturbances are present.

Reviewing Soft Constraints

Click **Soft Constraints** to see a list of soft constraints -- in this case an upper and lower bound on each OV.

Soft Constraints

ECR Parameters

This test evaluates the constraint ECR parameters to help you achieve the proper balance of using hard and soft constraints. If a constraint is too soft, an unacceptable violation may occur. If it is too hard, the controller might pay it too much attention. Moreover, making a constraint harder cannot prevent a violation if the constraint is fundamentally infeasible.

You have defined 8 soft constraints. The table below lists these and shows potential violations based on specified variable bounds and other factors.

Impact Factor: the increase in the MPC cost function caused by this constraint violation relative to the average such increase. Rows are sorted in order of decreasing impact.

Sensitivity Ratio: the increase in the MPC cost function caused by this constraint violation relative to the typical cost function magnitude when there are no violations.

We consider a possible constraint violation equal to 10% of the nominal OV range. It then estimates the impact of such a violation on the MPC objective function relative to the impact of other violations. A large impact factor indicates a high-priority controller objective, and vice versa.

Constraint	Assumed Violation	Impact Factor	Sensitivity Ratio
Lower limit: P	20	1509	1000
Upper limit: P	20	1509	1000
Lower limit: FSI	0.7	1.849	1.225
Upper limit: FSI	0.7	1.849	1.225
Lower limit: WI	0.2	0.1509	0.1
Upper limit: WI	0.2	0.1509	0.1
Lower limit: HHV	0.15	0.08491	0.05625
Upper limit: HHV	0.15	0.08491	0.05625

A sensitivity ratio greater than 1e+08 may degrade QP solution accuracy.

The Impact Factor column shows that using the default MinECR and MaxECR values give the pressure (P) a much higher priority than the other OVs. If we want the priorities to be more comparable, we should increase the pressure constraint ECR values and adjust the others too. For example, we consider

```
Obj.OV(1).MinECR = 0.5;
Obj.OV(1).MaxECR = 0.5;
Obj.OV(3).MinECR = 3;
Obj.OV(3).MaxECR = 3;
Obj.OV(4).MinECR = 80;
Obj.OV(4).MaxECR = 80;
```

Review the impact of this design change.

```
review(Obj)
```

Constraint	Assumed Violation	Impact Factor	Sensitivity Ratio
Lower limit: HHV	0.15	1.539	0.225
Upper limit: HHV	0.15	1.539	0.225
Lower limit: P	20	1.069	0.1563
Upper limit: P	20	1.069	0.1563
Lower limit: FSI	0.7	0.9311	0.1361
Upper limit: FSI	0.7	0.9311	0.1361
Lower limit: WI	0.2	0.6841	0.1
Upper limit: WI	0.2	0.6841	0.1

Notice from the Sensitivity Ratio column that all the sensitivity ratios are now less than unity. This means that the soft constraints will receive less attention than other terms in the MPC objective function, such as deviations of the MVs from their target values. Thus, it is likely that an output constraint violation would occur.

In order to give the output constraints higher priority than other MPC objectives, increase the Weights.ECR parameter from default 1e5 to a higher value to harden all the soft OV constraints.

```
Obj.Weights.ECR = 1e8;
```

Review the impact of this design change.

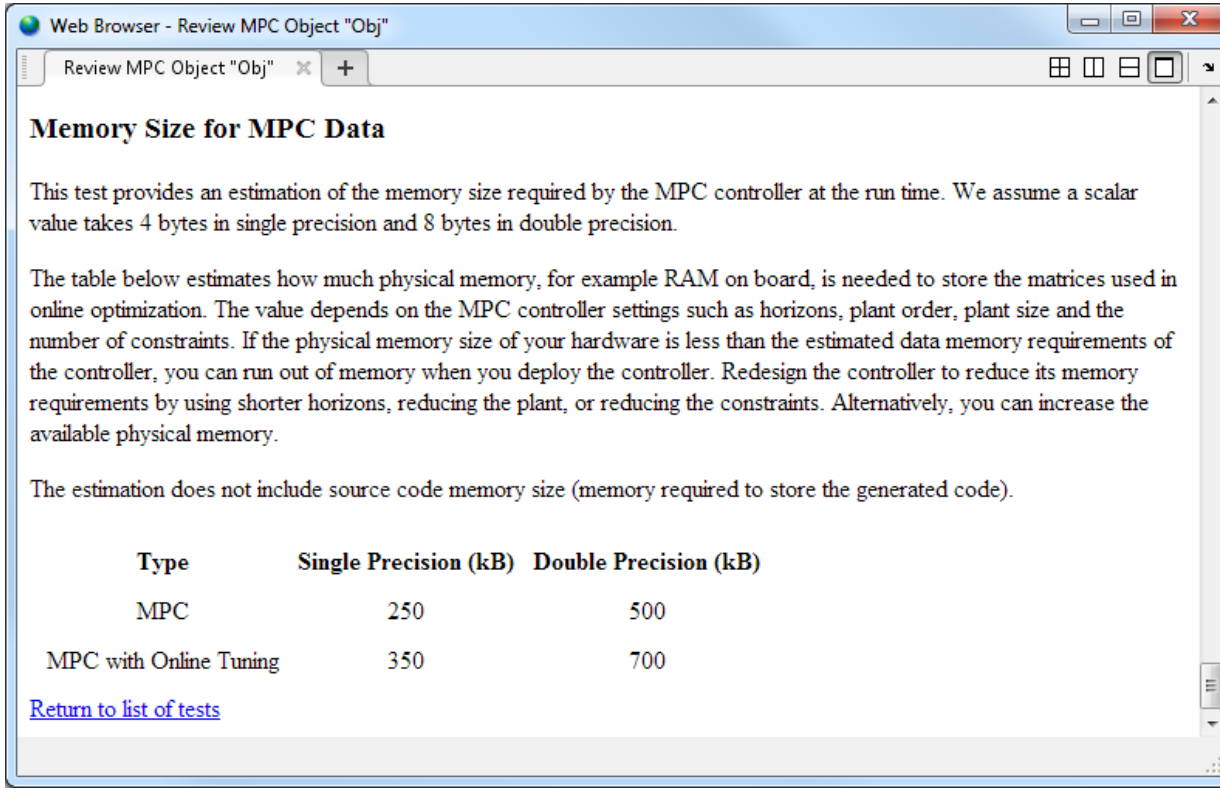
```
review(Obj)
```

Constraint	Assumed Violation	Impact Factor	Sensitivity Ratio
Lower limit: HHV	0.15	1.539	225
Upper limit: HHV	0.15	1.539	225
Lower limit: P	20	1.069	156.3
Upper limit: P	20	1.069	156.3
Lower limit: FSI	0.7	0.9311	136.1
Upper limit: FSI	0.7	0.9311	136.1
Lower limit: WI	0.2	0.6841	100
Upper limit: WI	0.2	0.6841	100

The controller is now a factor of 100 more sensitive to output constraint violations than to errors in target tracking.

Reviewing Data Memory Size

Click **Memory Size for MPC Data** to see the estimated memory size needed to store the MPC data matrices used on the hardware.



In this example, if the controller is running using single precision, it requires 250 KB of memory to store its matrices. If the controller memory size exceeds the memory available on the target system, you must redesign the controller to reduce its memory requirements. Alternatively, increase the memory available on the target system.

```
mpcverbosity(MPC_verbosity);
[~, hWebBrowser] = web;
close(hWebBrowser);
```

Bibliography

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

[2] Rawlings, J. B., and David Q. Mayne "Model Predictive Control: Theory and Design" Nob Hill Publishing, 2010.

Adaptive MPC Design

Adaptive MPC

In this section...

When to Use Adaptive MPC

MPC control predicts future behavior using a linear-time-invariant (LTI) dynamic model. In practice, such predictions are never exact, and a key tuning objective is to make MPC insensitive to prediction errors. In many applications, this approach is sufficient for robust controller performance.

If the plant is strongly nonlinear or its characteristics vary dramatically with time, LTI prediction accuracy might degrade so much that MPC performance becomes unacceptable. Adaptive MPC can address this degradation by adapting the prediction model for changing operating conditions. As implemented in the Model Predictive Control Toolbox software, adaptive MPC uses a fixed model structure, but allows the model's parameters to evolve with time. Ideally, whenever the controller requires a prediction (at the beginning of each control interval) it uses a model appropriate for the current conditions.

After you design an MPC controller for the average or most likely operating conditions of your control system, you can implement an adaptive MPC controller based on that design. For information about designing that initial controller, see "Controller Creation".

An alternative option for controlling a nonlinear or time-varying plant is to use gain-scheduled MPC control. See "Gain-Scheduled MPC".)

Plant Model

The plant model used as the basis for Adaptive MPC must be an LTI discrete-time, state-space model. See "Basic Models" in the Control System Toolbox documentation or "Linearization Basics" in the Simulink Control Design documentation for information about creating and modifying such systems. The plant model structure is as follows:

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

Here, the matrices A_p, B_{pu}, B_{pv}, B_{pd}, C_p, D_{pv} and D_{pd} are the parameters that can vary with time. The other variables in the expression are:

- k — Time index (current control interval).
- x_p — n_{xp} plant model states.
- u — n_u manipulated inputs (MVs). These are the one or more inputs that are adjusted by the MPC controller.
- v — n_v measured disturbance inputs.
- d — n_d unmeasured disturbance inputs.
- y — n_y plant outputs, including n_{ym} measured and n_{yu} unmeasured outputs. The total number of outputs, $n_y = n_{ym} + n_{yu}$. Also, $n_{ym} \geq 1$ (there is at least one measured output).

Additional requirements for the plant model in adaptive MPC control are:

- Sample time (`Ts`) is a constant and identical to the MPC control interval.
- Time delay (if any) is absorbed as discrete states (see, e.g., the Control System Toolbox `absorbDelay` command).
- n_{xp}, n_u, n_y, n_d, n_{ym}, and n_{yu} are all constants.
- Adaptive MPC prohibits direct feed-through from any manipulated variable to any plant output. Thus, $D_u = 0$ in the above model.

When you create the plant model, you specify it as an LTI state-space model with parameters A_p, B_p, C_p, D_p, and T_s. The sampling time is $T_s > 0$. The `InputGroup` and `OutputGroup` properties of the model designate input and output signal types (such as manipulated or measured). Each column in B_p and D_p represents a particular plant input variable. Grouping these columns according to input signal type yields the matrices B_{pu}, B_{pv}, B_{pd}, D_{pv} and D_{pd}. Similarly, each row in C_p, and D_p represent a particular output variable. Once you create these column and row assignments, you cannot change them as the system evolves in time.

For more details about creation of plant models for MPC control, see “Plant Specification”.

Nominal Operating Point

A traditional MPC controller includes a nominal operating point at which the plant model applies, such as the condition at which you linearize a nonlinear model to obtain the LTI approximation. The `Model.Nominal` property of the controller contains this information.

In adaptive MPC, as time evolves you should update the nominal operating point to be consistent with the updated plant model.

You can write the plant model in terms of deviations from the nominal conditions:

$$x_p(k+1) = \bar{x}_p + A_p\left(x_p(k) - \bar{x}_p\right) + B_p\left(u_t(k) - \bar{u}_t\right) + \overline{\Delta x}_p$$
$$y(k) = \bar{y} + C_p\left(x_p(k) - \bar{x}_p\right) + D_p\left(u_t(k) - \bar{u}_t\right).$$

Here, the matrices A_p, B_p, C_p, and D_p are the parameter matrices to be updated. u_t is the combined plant input variable, comprising the u, v, and d variables defined above. The nominal conditions to be updated are:

- $\bar{x}_p$ — n_{xp} nominal states
- $\overline{\Delta x}_p$ — n_{xp} nominal state increments
- $\bar{u}_t$ — n_{ut} nominal inputs
- $\bar{y}$ — n_y nominal outputs

State Estimation

By default, MPC uses a static Kalman filter (KF) to update its controller states, which include the n_{xp} plant model states, n_d (≥ 0) disturbance model states, and n_n (≥ 0) measurement noise model states. This KF requires two gain matrices, L and M. By default, the MPC controller calculates them during initialization. They depend upon the plant, disturbance, and noise model parameters, and assumptions regarding the stochastic noise signals driving the disturbance and noise models. For more details about state estimation in traditional MPC, see “Controller State Estimation”.

Adaptive MPC uses a Kalman filter by default, and adjusts the gains, L and M, at each control interval to maintain consistency with the updated plant model. The result is a linear-time-varying Kalman filter (LTVKF):

$$L_k = \left(A_k P_{k|k-1} C_{m,k}^T + N\right)\left(C_{m,k} P_{k|k-1} C_{m,k}^T + R\right)^{-1}$$

$$M_k = P_{k|k-1} C_{m,k}^T \left(C_{m,k} P_{k|k-1} C_{m,k}^T + R\right)^{-1}$$

$$P_{k+1|k} = A_k P_{k|k-1} A_k^T - \left(A_k P_{k|k-1} C_{m,k}^T + N\right) L_k^T + Q.$$

Here, Q, R, and N are constant covariance matrices defined as in MPC state estimation. A_k and $C_{m,k}$ are state-space parameter matrices for the entire controller state, defined as for traditional MPC but with the portions affected by the plant model updated to time k. The value $P_{k|k-1}$ is the state estimate error covariance matrix at time k based on information available at time k–1. Finally, L_k and M_k are the updated KF gain matrices. For details on the KF formulation used in traditional MPC, see "Controller State Estimation". By default, the initial condition, $P_{0|-1}$, is the static KF solution prior to any model updates.

The KF gain and the state error covariance matrix depend upon the model parameters and the assumptions leading to the constant Q, R, and N matrices. If the plant model is constant, the expressions for L_k and M_k converge to the equivalent static KF solution used in traditional MPC.

The equations for the controller state evolution at time k are identical to the KF formulation of traditional MPC described in "Controller State Estimation", but with the estimator gains and state space matrices updated to time k.

You have the option to update the controller state using a procedure external to the MPC controller, and then supply the updated state to MPC at each control instant, k. In this case, the MPC controller skips all KF and LTVKF calculations.

Related Examples

- "Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization"
- "Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation"

More About

- "Model Updating Strategy" on page 5-6
- "Controller State Estimation"

Model Updating Strategy

In this section...

Overview

Typically, to implement adaptive MPC control, you employ one of the following model-updating strategies:

Successive linearization. Given a mechanistic plant model, e.g., a set of nonlinear ordinary differential and algebraic equations, derive its LTI approximation at the current operating condition. For example, Simulink Control Design software provides linearization tools for this purpose.

Using a Linear Parameter Varying (LPV) model. Control System Toolbox software provides a LPV System Simulink block that allows you to specify an array of LTI models with scheduling parameters. You can perform batch linearization offline to obtain an array of plant models at the desired operating points and then use them in the LPV System block to provide model updating to the Adaptive MPC Controller Simulink block.

Online parameter estimation. Given an empirical model structure and initial estimates of its parameters, use the available real-time plant measurements to estimate the current model parameters. For example, the System Identification Toolbox™ software provides real-time parameter estimation tools.

Other Considerations

There are several factors to keep in mind when designing and implementing an adaptive MPC controller.

- Before attempting adaptive MPC, define and tune an MPC controller for the most typical (nominal) operating condition. Make sure the system can tolerate some prediction error. Test this tolerance via simulations in which the MPC prediction model differs from the plant. See "MPC Design".
- An adaptive MPC controller requires more real-time computations than traditional MPC. In addition to the state estimation calculation, you must also implement and test a model-updating strategy, which might be computationally intensive.

- You must determine MPC tuning constants that provide robust performance over the expected range of model parameters. See “Tuning Weights”.
- Model updating via online parameter estimation is most effective when parameter variations occur gradually.
- When implementing adaptive MPC control, adapt only parameters defining the `Model.Plant` property of the controller. The disturbance and noise models, if any, remain constant.

See Also

Adaptive MPC Controller

Related Examples

- “Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization”
- “Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation”

More About

- “Adaptive MPC” on page 5-2

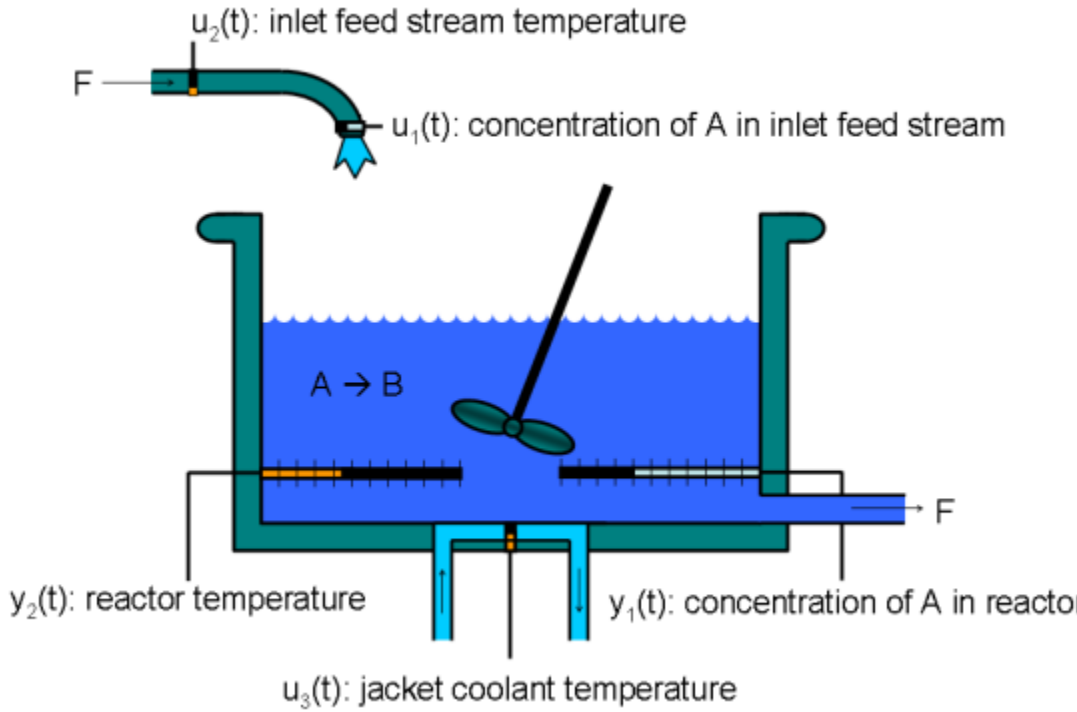
Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization

This example shows how to use an Adaptive MPC controller to control a nonlinear continuous stirred tank reactor (CSTR) as it transitions from low conversion rate to high conversion rate.

A first principle nonlinear plant model is available and being linearized at each control interval. The adaptive MPC controller then updates its internal predictive model with the linearized plant model and achieves nonlinear control successfully.

About the Continuous Stirred Tank Reactor

A Continuously Stirred Tank Reactor (CSTR) is a common chemical system in the process industry. A schematic of the CSTR system is:



This is a jacketed non-adiabatic tank reactor described extensively in Seborg's book, "Process Dynamics and Control", published by Wiley, 2004. The vessel is assumed to be perfectly mixed, and a single first-order exothermic and irreversible reaction, A --> B, takes place. The inlet stream of reagent A is fed to the tank at a constant volumetric rate. The product stream exits continuously at the same volumetric rate and liquid density is constant. Thus the volume of reacting liquid is constant.

The inputs of the CSTR model are:

$u_1 = CA_i$ Concentration of A in inlet feed stream$[kgmol/m^3]$
$u_2 = T_i$ Inlet feed stream temperature$[K]$
$u_3 = T_c$ Jacket coolant temperature$[K]$

and the outputs (y(t)), which are also the states of the model (x(t)), are:

$y_1 = x_1 = CA$ Concentration of A in reactor tank$[kgmol/m^3]$
$y_2 = x_2 = T$ Reactor temperature$[K]$

The control objective is to maintain the concentration of reagent A, CA at its desired setpoint, which changes over time when reactor transitions from low conversion rate to high conversion rate. The coolant temperature T_c is the manipulated variable used by the MPC controller to track the reference as well as reject the measured disturbance arising from the inlet feed stream temperature T_i. The inlet feed stream concentration, CA_i, is assumed to be constant. The Simulink model `mpc_cstr_plant` implements the nonlinear CSTR plant.

We also assume that direct measurements of concentrations are unavailable or infrequent, which is the usual case in practice. Instead, we use a "soft sensor" to estimate CA based on temperature measurements and the plant model.

About Adaptive Model Predictive Control

It is well known that the CSTR dynamics are strongly nonlinear with respect to reactor temperature variations and can be open-loop unstable during the transition from one operating condition to another. A single MPC controller designed at a particular operating condition cannot give satisfactory control performance over a wide operating range.

To control the nonlinear CSTR plant with linear MPC control technique, you have a few options:

- If a linear plant model cannot be obtained at run time, first you need to obtain several linear plant models offline at different operating conditions that cover the typical operating range. Next you can choose one of the two approaches to implement MPC control strategy:

(1) Design several MPC controllers offline, one for each plant model. At run time, use Multiple MPC Controller block that switches MPC controllers from one to another based on a desired scheduling strategy. See "Gain Scheduled MPC Control of Nonlinear Chemical Reactor" for more details. Use this approach when the plant models have different orders or time delays.

(2) Design one MPC controller offline at the initial operating point. At run time, use Adaptive MPC Controller block (updating predictive model at each control interval) together with Linear Parameter Varying (LPV) System block (supplying linear plant model with a scheduling strategy). See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Linear Parameter Varying System" for more details. Use this approach when all the plant models have the same order and time delay.

- If a linear plant model can be obtained at run time, you should use Adaptive MPC Controller block to achieve nonlinear control. There are two typical ways to obtain a linear plant model online:

(1) Use successive linearization as shown in this example. Use this approach when a nonlinear plant model is available and can be linearized at run time.

(2) Use online estimation to identify a linear model when loop is closed. See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation" for more details. Use this approach when linear plant model cannot be obtained from either an LPV system or successive linearization.

Obtain Linear Plant Model at Initial Operating Condition

To linearize the plant, Simulink® and Simulink Control Design® are required.

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

To implement an adaptive MPC controller, first you need to design a MPC controller at the initial operating point where CAi is 10 kgmol/m^3, Ti and Tc are 298.15 K.

Create operating point specification.

```
plant_mdl = 'mpc_cstr_plant';
op = operspec(plant_mdl);
```

Feed concentration is known at the initial condition.

```
op.Inputs(1).u = 10;
op.Inputs(1).Known = true;
```

Feed temperature is known at the initial condition.

```
op.Inputs(2).u = 298.15;
op.Inputs(2).Known = true;
```

Coolant temperature is known at the initial condition.

```
op.Inputs(3).u = 298.15;
op.Inputs(3).Known = true;
```

Compute initial condition.

```
[op_point, op_report] = findop(plant_mdl,op);
```

```
 Operating Point Search Report:
---------------------------------

 Operating Report for the Model mpc_cstr_plant.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) mpc_cstr_plant/CSTR/Integrator
      x:           311      dx:      8.12e-11 (0)
(2.) mpc_cstr_plant/CSTR/Integrator1
      x:          8.57      dx:     -6.87e-12 (0)

Inputs:
----------
(1.) mpc_cstr_plant/CAi
```

```
      u:            10
(2.) mpc_cstr_plant/Ti
      u:           298
(3.) mpc_cstr_plant/Tc
      u:           298

Outputs:
----------
(1.) mpc_cstr_plant/T
      y:           311    [-Inf Inf]
(2.) mpc_cstr_plant/CA
      y:          8.57    [-Inf Inf]
```

Obtain nominal values of x, y and u.

```
x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];
```

Obtain linear plant model at the initial condition.

```
sys = linearize(plant_mdl, op_point);
```

Drop the first plant input CAi because it is not used by MPC.

```
sys = sys(:,2:3);
```

Discretize the plant model because Adaptive MPC controller only accepts a discrete-time plant model.

```
Ts = 0.5;
plant = c2d(sys,Ts);
```

Design MPC Controller

You design an MPC at the initial operating condition. When running in the adaptive mode, the plant model is updated at run time.

Specify signal types used in MPC.

```
plant.InputGroup.MeasuredDisturbances = 1;
plant.InputGroup.ManipulatedVariables = 2;
plant.OutputGroup.Measured = 1;
```

```
plant.OutputGroup.Unmeasured = 2;
plant.InputName = {'Ti','Tc'};
plant.OutputName = {'T','CA'};
```

Create MPC controller with default prediction and control horizons

```
mpcobj = mpc(plant);
```

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
   for output(s) y1 and zero weight for output(s) y2
```

Set nominal values in the controller

```
mpcobj.Model.Nominal = struct('X', x0, 'U', u0(2:3), 'Y', y0, 'DX', [0 0]);
```

Set scale factors because plant input and output signals have different orders of magnitude

```
Uscale = [30 50];
Yscale = [50 10];
mpcobj.DV(1).ScaleFactor = Uscale(1);
mpcobj.MV(1).ScaleFactor = Uscale(2);
mpcobj.OV(1).ScaleFactor = Yscale(1);
mpcobj.OV(2).ScaleFactor = Yscale(2);
```

Let reactor temperature T float (i.e. with no setpoint tracking error penalty), because the objective is to control reactor concentration CA and only one manipulated variable (coolant temperature Tc) is available.

```
mpcobj.Weights.OV = [0 1];
```

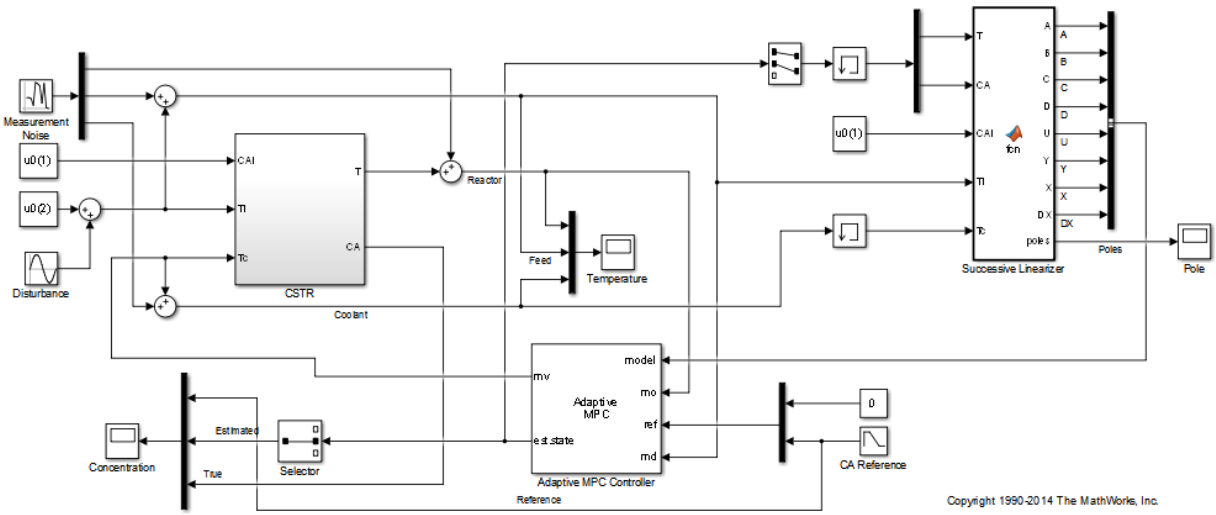
Due to the physical constraint of coolant jacket, Tc rate of change is bounded by degrees per minute.

```
mpcobj.MV.RateMin = -2;
mpcobj.MV.RateMax = 2;
```

Implement Adaptive MPC Control of CSTR Plant in Simulink (R)

Open the Simulink model.

```
mdl = 'ampc_cstr_linearization';
open_system(mdl);
```



The model includes three parts:

1. The "CSTR" block implements the nonlinear plant model.
2. The "Adaptive MPC Controller" block runs the designed MPC controller in the adaptive mode.
3. The "Successive Linearizer" block in a MATLAB Function block that linearizes a first principle nonlinear CSTR plant and provides the linear plant model to the "Adaptive MPC Controller" block at each control interval. Double click the block to see the MATLAB code. You can use the block as a template to develop appropriate linearizer for your own applications.

Note that the new linear plant model must be a discrete time state space system with the same order and sample time as the original plant model has. If the plant has time delay, it must also be same as the original time delay and absorbed into the state space model.

Validate Adaptive MPC Control Performance

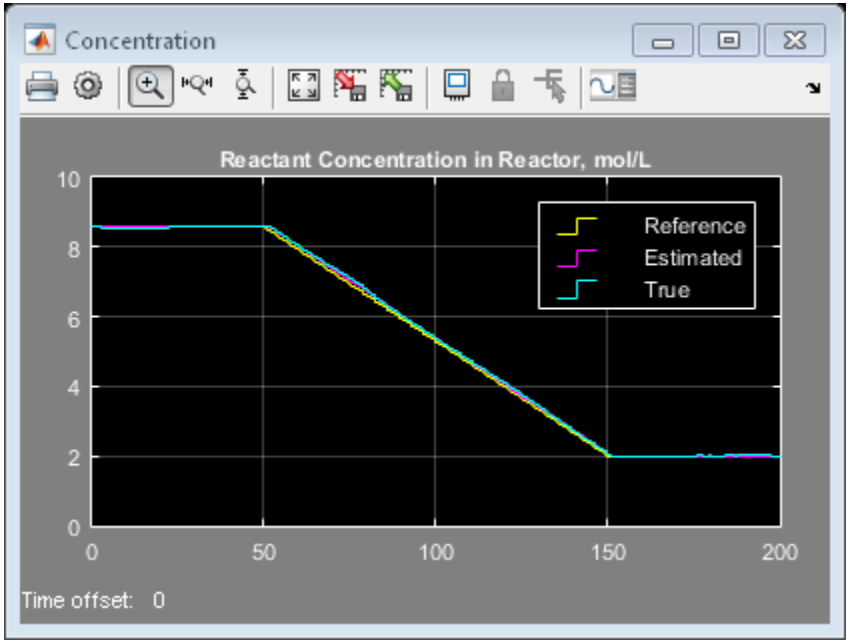
Controller performance is validated against both setpoint tracking and disturbance rejection.

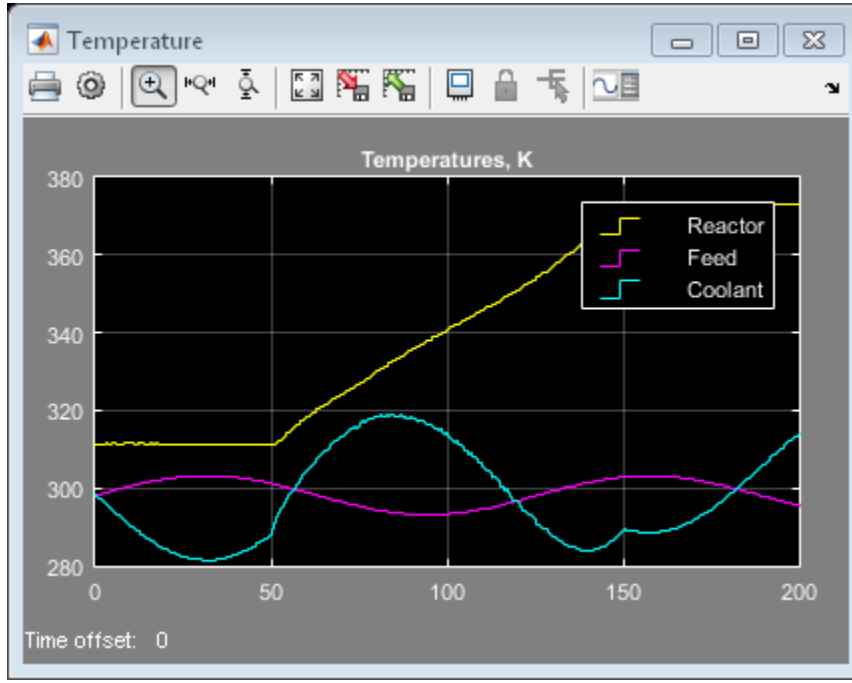
- Tracking: reactor concentration CA setpoint transitions from original 8.57 (low conversion rate) to 2 (high conversion rate) kgmol/m^3. During the transition, the plant first becomes unstable then stable again (see the poles plot).
- Regulating: feed temperature Ti has slow fluctuation represented by a sine wave with amplitude of 5 degrees, which is a measured disturbance fed to the MPC controller.

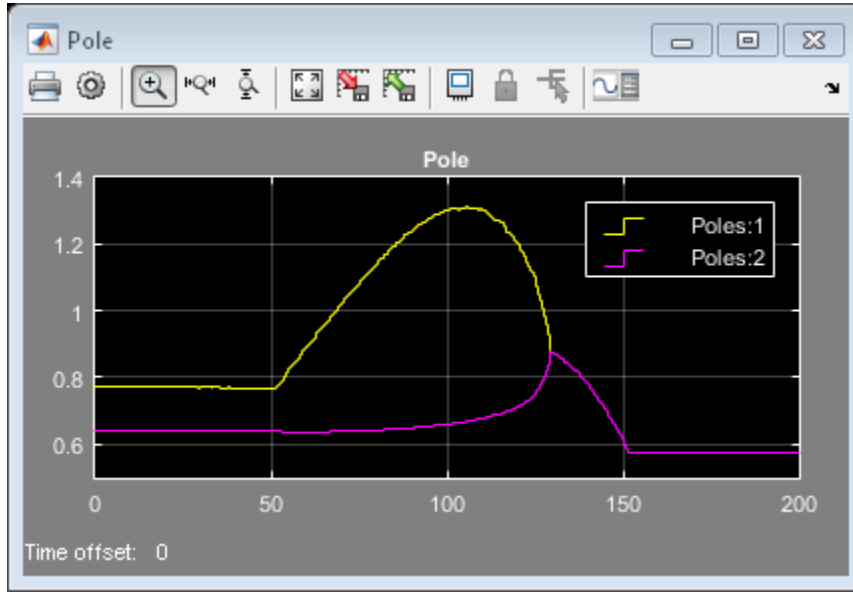
Simulate the closed-loop performance.

```
open_system([mdl '/Concentration'])
open_system([mdl '/Temperature'])
open_system([mdl '/Pole'])
sim(mdl);
```

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




Temperature
Temperatures, K
380
360
340
320
300
280
0
50
100
150
200
Reactor
Feed
Coolant
Time offset: 0

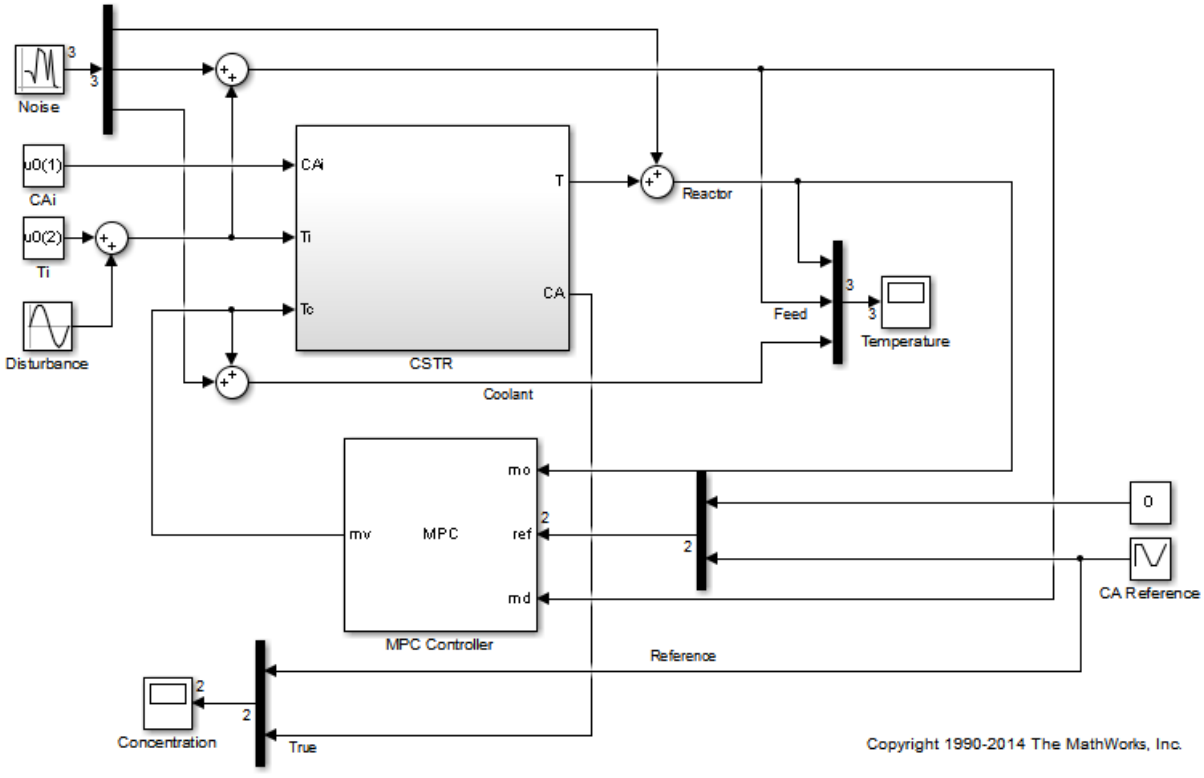


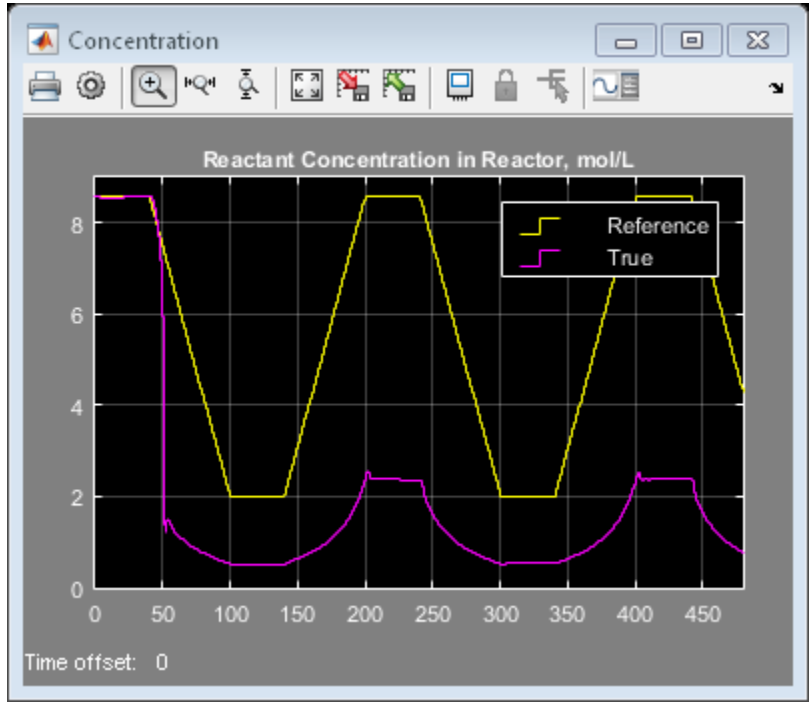
The tracking and regulating performance is very satisfactory. In an application to a real reactor, however, model inaccuracies and unmeasured disturbances could cause poorer tracking than shown here. Additional simulations could be used to study these effects.

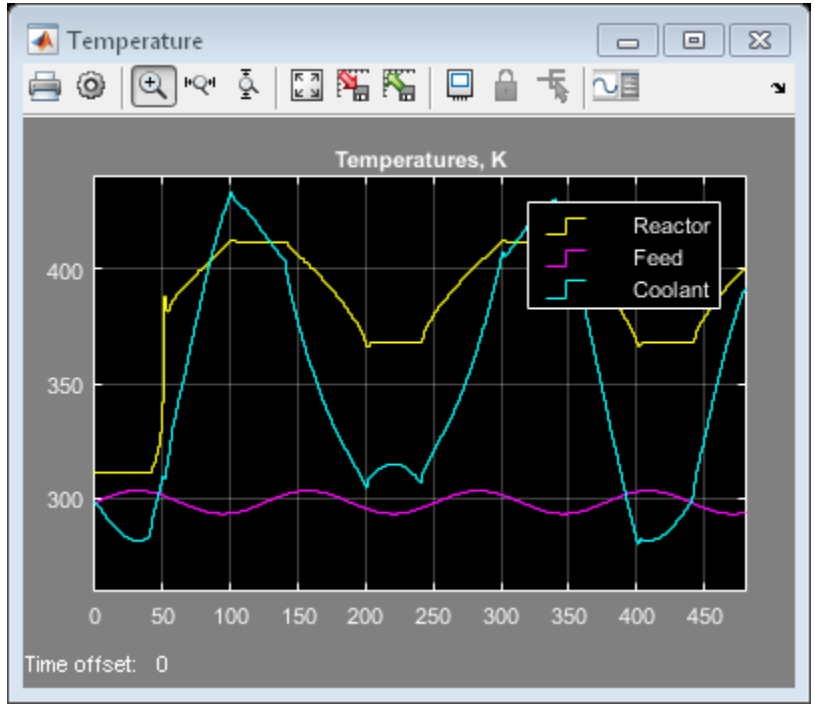
Compare with Non-Adaptive MPC Control

Adaptive MPC provides superior control performance than a non-adaptive MPC. To illustrate this point, the control performance of the same MPC controller running in the non-adaptive mode is shown below. The controller is implemented with a MPC Controller block.

```
mdl1 = 'ampc_cstr_no_linearization';
open_system(mdl1);
open_system([mdl1 '/Concentration'])
open_system([mdl1 '/Temperature'])
sim(mdl1);
```


Noise
3
3
u0(1)
CAi
u0(2)
Ti
Disturbance
CAi
Ti
Tc
CA
T
CSTR
Reactor
Feed
3
3
Temperature
Coolant
mo
mv
MPC
ref
md
2
2
MPC Controller
0
CA Reference
Reference
2
2
Concentration
True
Copyright 1990-2014 The MathWorks, Inc.


Concentration
Reactant Concentration in Reactor, mol/L
Reference
True
8
6
4
2
0
0
50
100
150
200
250
300
350
400
450
Time offset: 0



As expected, the tracking and regulating performance is unacceptable.

```
bdclose(mdl)
bdclose(mdl1)
```

See Also

Adaptive MPC Controller

Related Examples

- "Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation"

More About

- "Adaptive MPC" on page 5-2

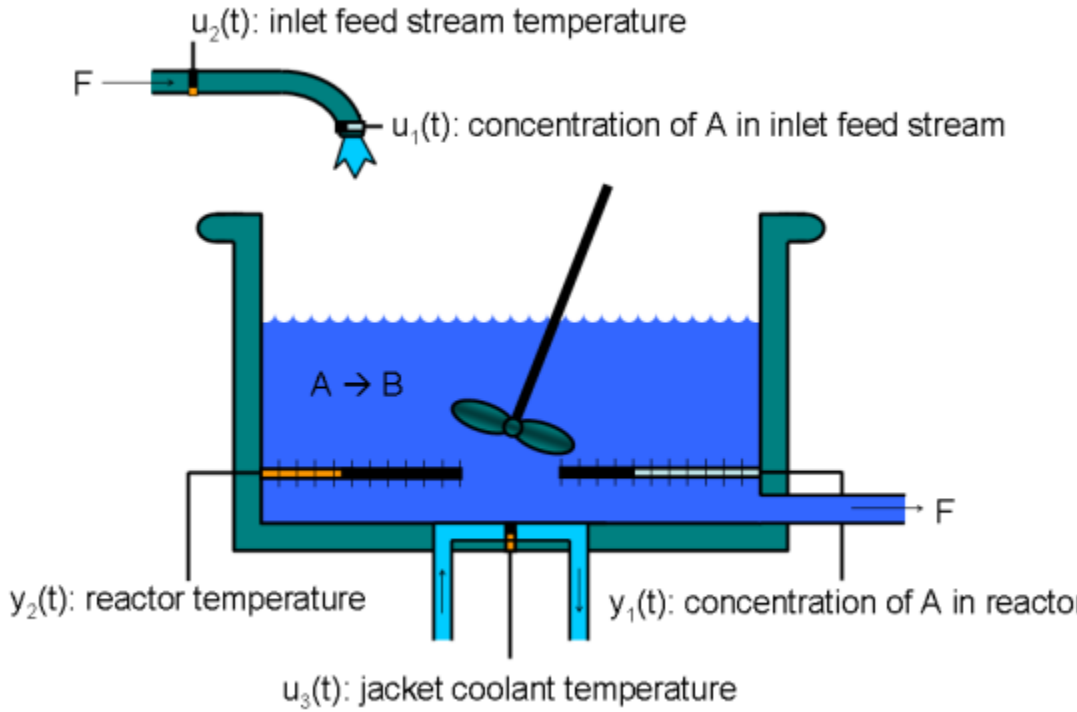
Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation

This example shows how to use an Adaptive MPC controller to control a nonlinear continuous stirred tank reactor (CSTR) as it transitions from low conversion rate to high conversion rate.

A discrete time ARX model is being identified online by the Recursive Polynomial Model Estimator block at each control interval. The adaptive MPC controller uses it to update internal plant model and achieves nonlinear control successfully.

About the Continuous Stirred Tank Reactor

A Continuously Stirred Tank Reactor (CSTR) is a common chemical system in the process industry. A schematic of the CSTR system is:



This is a jacketed non-adiabatic tank reactor described extensively in Seborg's book, "Process Dynamics and Control", published by Wiley, 2004. The vessel is assumed to be perfectly mixed, and a single first-order exothermic and irreversible reaction, A --> B, takes place. The inlet stream of reagent A is fed to the tank at a constant volumetric rate. The product stream exits continuously at the same volumetric rate and liquid density is constant. Thus the volume of reacting liquid is constant.

The inputs of the CSTR model are:

$$\begin{array}{ll} u_1 = CA_i & \text{Concentration of A in inlet feed stream}[kgmol/m^3] \\ u_2 = T_i & \text{Inlet feed stream temperature}[K] \\ u_3 = T_c & \text{Jacket coolant temperature}[K] \end{array}$$

and the outputs (y(t)), which are also the states of the model (x(t)), are:

$$\begin{array}{ll} y_1 = x_1 = CA & \text{Concentration of A in reactor tank}[kgmol/m^3] \\ y_2 = x_2 = T & \text{Reactor temperature}[K] \end{array}$$

The control objective is to maintain the reactor temperature T at its desired setpoint, which changes over time when reactor transitions from low conversion rate to high conversion rate. The coolant temperature T_c is the manipulated variable used by the MPC controller to track the reference as well as reject the measured disturbance arising from the inlet feed stream temperature T_i. The inlet feed stream concentration, CA_i, is assumed to be constant. The Simulink model `mpc_cstr_plant` implements the nonlinear CSTR plant.

About Adaptive Model Predictive Control

It is well known that the CSTR dynamics are strongly nonlinear with respect to reactor temperature variations and can be open-loop unstable during the transition from one operating condition to another. A single MPC controller designed at a particular operating condition cannot give satisfactory control performance over a wide operating range.

To control the nonlinear CSTR plant with linear MPC control technique, you have a few options:

- If a linear plant model cannot be obtained at run time, first you need to obtain several linear plant models offline at different operating conditions that cover the typical

operating range. Next you can choose one of the two approaches to implement MPC control strategy:

(1) Design several MPC controllers offline, one for each plant model. At run time, use Multiple MPC Controller block that switches MPC controllers from one to another based on a desired scheduling strategy. See "Gain Scheduled MPC Control of Nonlinear Chemical Reactor" for more details. Use this approach when the plant models have different orders or time delays.

(2) Design one MPC controller offline at the initial operating point. At run time, use Adaptive MPC Controller block (updating predictive model at each control interval) together with Linear Parameter Varying (LPV) System block (supplying linear plant model with a scheduling strategy). See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Linear Parameter Varying System" for more details. Use this approach when all the plant models have the same order and time delay.

- If a linear plant model can be obtained at run time, you should use Adaptive MPC Controller block to achieve nonlinear control. There are two typical ways to obtain a linear plant model online:

(1) Use successive linearization. See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization" for more details. Use this approach when a nonlinear plant model is available and can be linearized at run time.

(2) Use online estimation to identify a linear model when loop is closed, as shown in this example. Use this approach when linear plant model cannot be obtained from either an LPV system or successive linearization.

Obtain Linear Plant Model at Initial Operating Condition

To linearize the plant, Simulink® and Simulink Control Design® are required.

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

To implement an adaptive MPC controller, first you need to design a MPC controller at the initial operating point where CAi is 10 kgmol/m^3, Ti and Tc are 298.15 K.

Create operating point specification.

```
plant_mdl = 'mpc_cstr_plant';
op = operspec(plant_mdl);
```

Feed concentration is known at the initial condition.

```
op.Inputs(1).u = 10;
op.Inputs(1).Known = true;
```

Feed temperature is known at the initial condition.

```
op.Inputs(2).u = 298.15;
op.Inputs(2).Known = true;
```

Coolant temperature is known at the initial condition.

```
op.Inputs(3).u = 298.15;
op.Inputs(3).Known = true;
```

Compute initial condition.

```
[op_point, op_report] = findop(plant_mdl,op);
```

```
 Operating Point Search Report:
---------------------------------

 Operating Report for the Model mpc_cstr_plant.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) mpc_cstr_plant/CSTR/Integrator
      x:           311      dx:      8.12e-11 (0)
(2.) mpc_cstr_plant/CSTR/Integrator1
      x:          8.57      dx:     -6.87e-12 (0)

Inputs:
----------
(1.) mpc_cstr_plant/CAi
      u:            10
(2.) mpc_cstr_plant/Ti
```

```
      u:           298
(3.) mpc_cstr_plant/Tc
      u:           298

Outputs:
----------
(1.) mpc_cstr_plant/T
      y:           311    [-Inf Inf]
(2.) mpc_cstr_plant/CA
      y:          8.57    [-Inf Inf]
```

Obtain nominal values of x, y and u.

```
x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];
```

Obtain linear plant model at the initial condition.

```
sys = linearize(plant_mdl, op_point);
```

Drop the first plant input CAi and second output CA because they are not used by MPC.

```
sys = sys(1,2:3);
```

Discretize the plant model because Adaptive MPC controller only accepts a discrete-time plant model.

```
Ts = 0.5;
plant = c2d(sys,Ts);
```

Design MPC Controller

You design an MPC at the initial operating condition. When running in the adaptive mode, the plant model is updated at run time.

Specify signal types used in MPC.

```
plant.InputGroup.MeasuredDisturbances = 1;
plant.InputGroup.ManipulatedVariables = 2;
plant.OutputGroup.Measured = 1;
plant.InputName = {'Ti','Tc'};
plant.OutputName = {'T'};
```

Create MPC controller with default prediction and control horizons

```
mpcobj = mpc(plant);
```

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

Set nominal values in the controller

```
mpcobj.Model.Nominal = struct('X', x0, 'U', u0(2:3), 'Y', y0(1), 'DX', [0 0]);
```

Set scale factors because plant input and output signals have different orders of magnitude

```
Uscale = [30 50];
Yscale = 50;
mpcobj.DV.ScaleFactor = Uscale(1);
mpcobj.MV.ScaleFactor = Uscale(2);
mpcobj.OV.ScaleFactor = Yscale;
```

Due to the physical constraint of coolant jacket, Tc rate of change is bounded by 2 degrees per minute.

```
mpcobj.MV.RateMin = -2;
mpcobj.MV.RateMax = 2;
```

Reactor concentration is not directly controlled in this example. If reactor temperature can be successfully controlled, the concentration will achieve desired performance requirement due to the strongly coupling between the two variables.

Implement Adaptive MPC Control of CSTR Plant in Simulink (R)

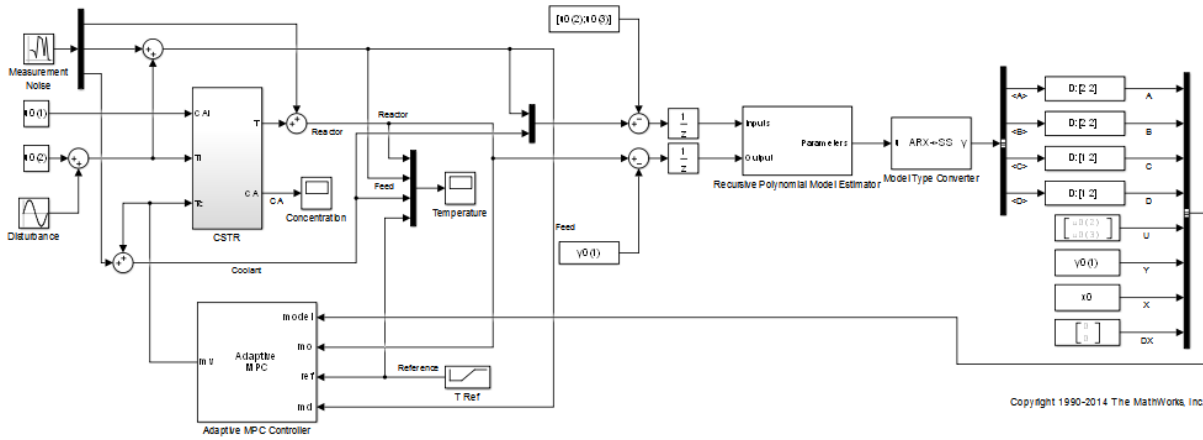
To run this example with online estimation, System Identification® is required.

```
if ~mpcchecktoolboxinstalled('ident')
    disp('System Identification(R) is required to run this example.')
    return
end
```

Open the Simulink model.

```
mdl = 'ampc_cstr_estimation';
```

```
open_system(mdl);
```



The model includes three parts:

1. The "CSTR" block implements the nonlinear plant model.
2. The "Adaptive MPC Controller" block runs the designed MPC controller in the adaptive mode.
3. The "Recursive Polynomial Model Estimator" block estimates a two-input (Ti and Tc) and one-output (T) discrete time ARX model based on the measured temperatures. The estimated model is then converted into state space form by the "Model Type Converter" block and fed to the "Adaptive MPC Controller" block at each control interval.

In this example, the initial plant model is used to initialize the online estimator with parameter covariance matrix set to 1. The online estimation method is "Kalman Filter" with noise covariance matrix set to 0.01. The online estimation result is sensitive to these parameters and you can further adjust them to achieve better estimation result.

Both "Recursive Polynomial Model Estimator" and "Model Type Converter" are provided by System Identification Toolbox. You can use the two blocks as a template to develop appropriate online model estimation for your own applications.

The initial value of A(q) and B(q) variables are populated with the numerator and denominator of the initial plant model.

```
[num, den] = tfdata(plant);
Aq = den{1};
Bq = num;
```

Note that the new linear plant model must be a discrete time state space system with the same order and sample time as the original plant model has. If the plant has time delay, it must also be same as the original time delay and absorbed into the state space model.

Validate Adaptive MPC Control Performance

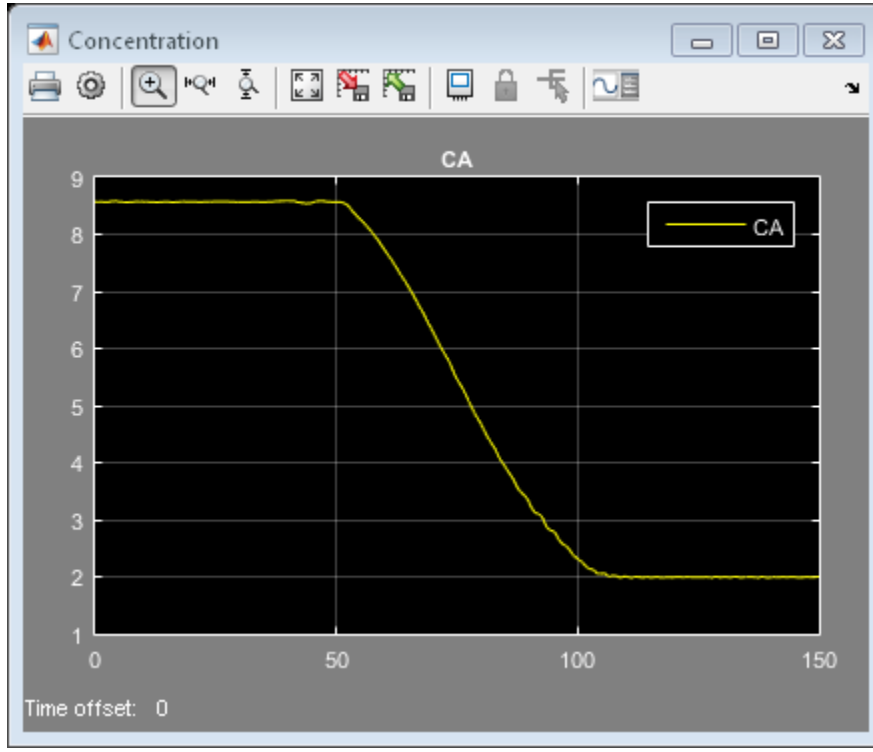
Controller performance is validated against both setpoint tracking and disturbance rejection.

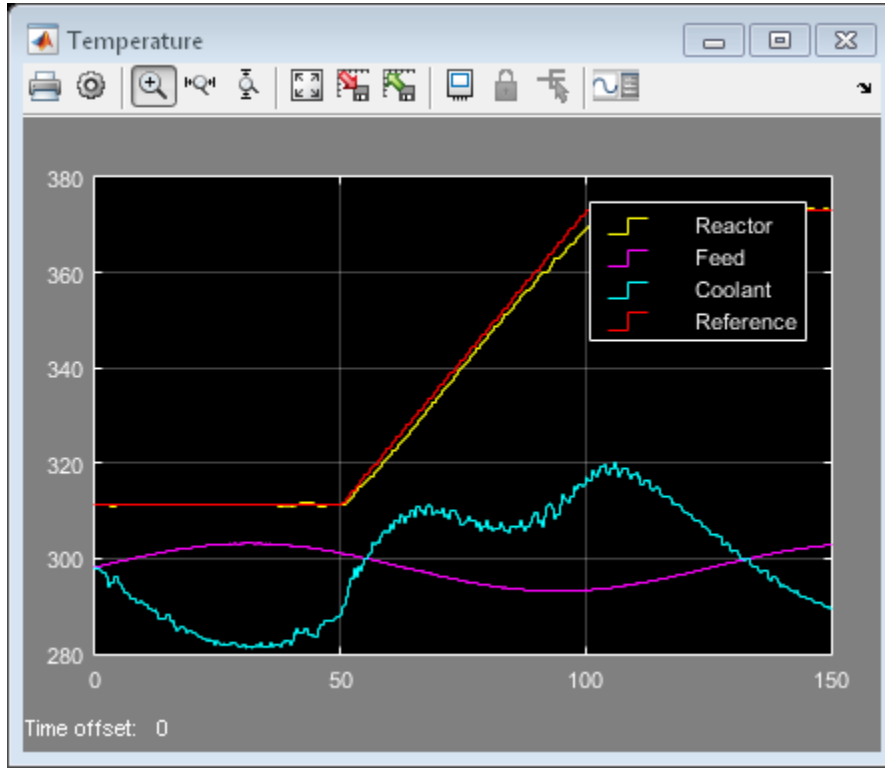
- Tracking: reactor temperature T setpoint transitions from original 311 K (low conversion rate) to 377 K (high conversion rate) kgmol/m^3. During the transition, the plant first becomes unstable then stable again (see the poles plot).
- Regulating: feed temperature Ti has slow fluctuation represented by a sine wave with amplitude of 5 degrees, which is a measured disturbance fed to MPC controller.

Simulate the closed-loop performance.

```
open_system([mdl '/Concentration'])
open_system([mdl '/Temperature'])
sim(mdl);
```

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


Concentration
CA
CA
9
8
7
6
5
4
3
2
1
0
50
100
150
Time offset: 0

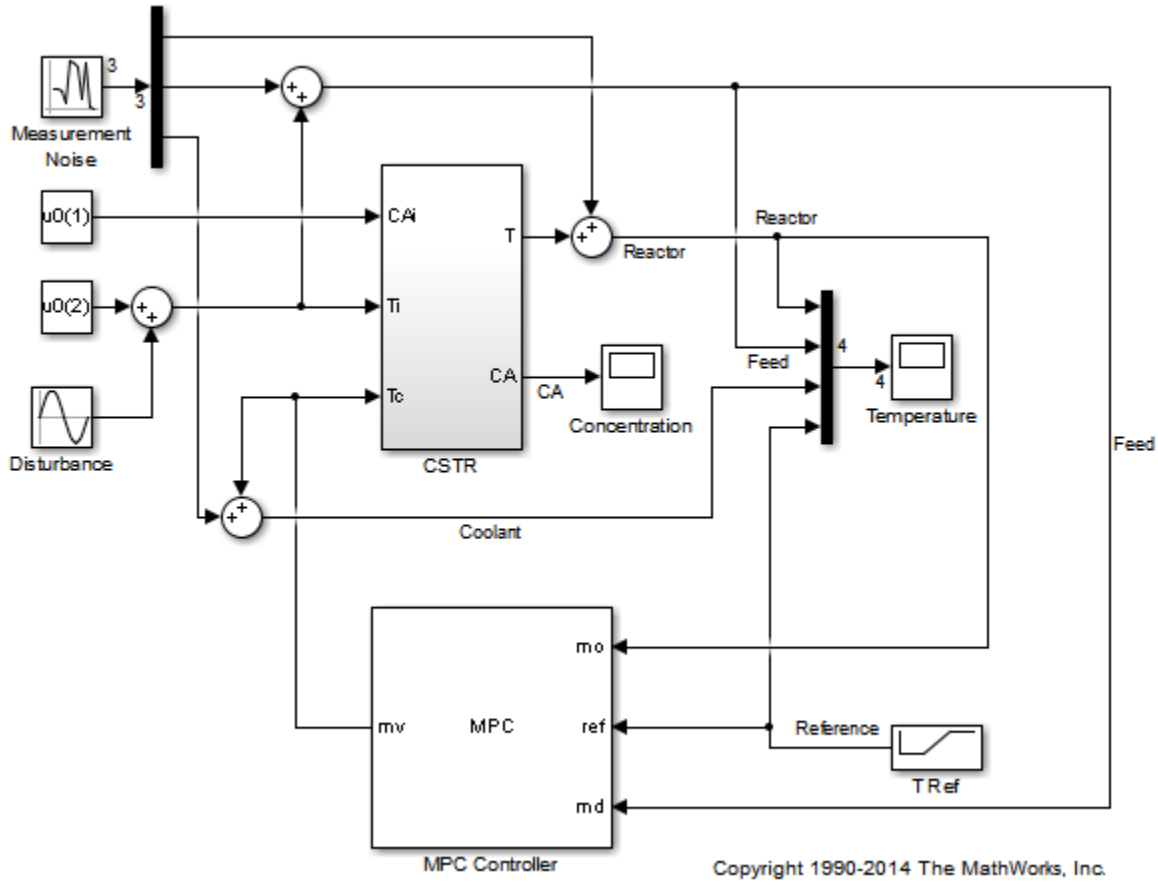


The tracking and regulating performance is very satisfactory.

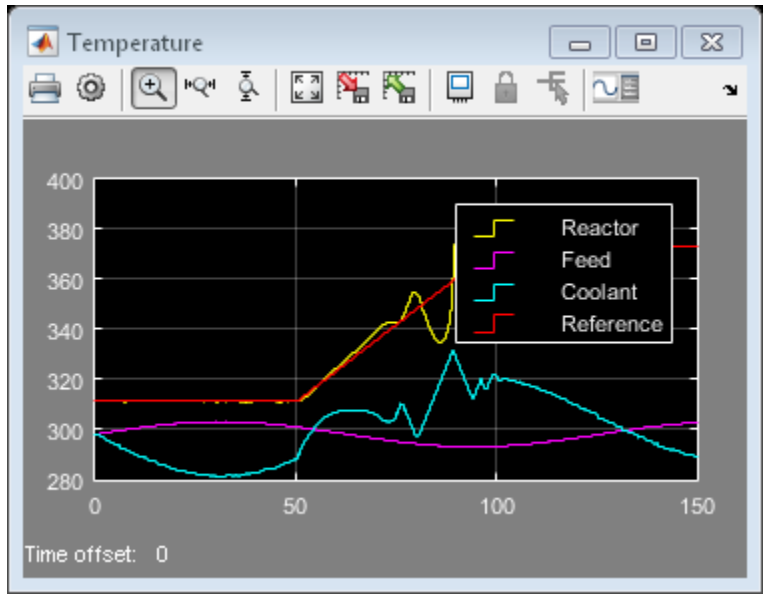
Compare with Non-Adaptive MPC Control

Adaptive MPC provides superior control performance than a non-adaptive MPC. To illustrate this point, the control performance of the same MPC controller running in the non-adaptive mode is shown below. The controller is implemented with a MPC Controller block.

```
mdl1 = 'ampc_cstr_no_estimation';
open_system(mdl1);
open_system([mdl1 '/Concentration'])
open_system([mdl1 '/Temperature'])
sim(mdl1);
```


3
3
Measurement
Noise
u0(1)
u0(2)
Disturbance
CAi
Ti
Tc
T
CA
CSTR
Reactor
Reactor
Feed
4
4
Temperature
CA
Concentration
Feed
Coolant
mo
mv
MPC
ref
md
Reference
TRef
MPC Controller
Copyright 1990-2014 The MathWorks, Inc.


Concentration
CA
CA
8
6
4
2
0
50
100
150
Time offset: 0


Temperature
400
380
360
340
320
300
280
Reactor
Feed
Coolant
Reference
0
50
100
150
Time offset: 0

As expected, the tracking and regulating performance is unacceptable.

```
bdclose(mdl)
bdclose(mdl1)
```

See Also

Adaptive MPC Controller

Related Examples

- "Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization"

More About

- "Adaptive MPC" on page 5-2

Explicit MPC Design

Explicit MPC

A traditional model predictive controller solves a quadratic program (QP) at each control interval to determine the optimal manipulated variable (MV) adjustments. Mathematically, these adjustments are an implicit nonlinear function of the current controller state and other variables, such as the current output reference values. Suppose that all the independent variables affecting the QP solution form a vector, x. The optimal MV adjustments are then $u = f(x)$, where f is the implicit function to be solved by quadratic programming. The Model Predictive Control Toolbox software imposes restrictions that force the QP to have a unique solution.

Evaluating the solution of $u = f(x)$ via QP can be time consuming, however, and the required time can vary significantly from one control interval to the next. In applications that require the solution to be obtained consistently within a certain elapsed time, which could be of order milliseconds or microseconds, the implicit MPC approach might be unsuitable.

As shown in "Optimization Problem", if x is such that none of the QP's inequality constraints are active at the solution, the MPC control law reduces to a linear function of x:

$$u = Fx + G.$$

Here, F and G are constants. Similarly, if x stays within a region for which a fixed subset of the inequality constraints are active, the QP solution is again a linear function of x but with different F and G constants.

Explicit MPC uses offline computations to determine all polyhedral regions in which the optimal MV adjustments are a linear function of x and the corresponding control-law constants. When the controller operates in real time, the explicit MPC controller performs the following steps at each control instant, k:

1. Uses the available measurements to estimate the controller state, as in traditional MPC.
2. Uses this estimated state and current values of the other independent variables to form $x(k)$.
3. Identifies the region in which $x(k)$ resides.
4. Looks up the predetermined F and G constants for this region.
5. Evaluates the linear function $u(k) = Fx(k) + G$.

You can establish a tight upper bound for the time required in each step. The total computational time can be quite small of the number of regions is not too large. In fact, the time required in step 3 dominates, and the memory needed to store all the linear control laws and polyhedral regions becomes excessive. The number of regions characterizing $u = f(x)$ depends primarily on the combinations of QP inequality constraint that could be active at the solution. Thus, an explicit MPC involving many constraints might require too much computational effort or memory. In that case, a traditional (implicit) implementation may be preferable.

Related Examples

- "Explicit MPC Control of a Single-Input-Single-Output Plant"
- "Explicit MPC Control of an Aircraft with Unstable Poles"
- "Explicit MPC Control of DC Servomotor with Constraint on Unmeasured Output"

More About

- "Design Workflow for Explicit MPC" on page 6-4

Design Workflow for Explicit MPC

In this section...

To create an explicit MPC controller, you must first design a traditional (implicit) MPC controller. You then generate an explicit MPC controller based on the traditional controller design.

Traditional (Implicit) MPC Design

First design a traditional (implicit) MPC for your application and test it in simulations. Key considerations are as follows:

- The Model Predictive Control Toolbox software currently supports the following as independent variables for explicit MPC:
  - n_{xc} controller state variables (plant, disturbance, and measurement noise model states).
  - n_y (≥ 1) output reference values, where n_y is the number of plant output variables.
  - n_v (≥ 0) measured plant disturbance signals.

  Thus, you must fix most MPC design parameters prior to determining an explicit MPC. Fixed parameters include prediction models (plant, disturbance and measurement noise), scale factors, horizons, penalty weights, manipulated variable targets, and constraint bounds.

  For information about designing a traditional MPC controller, see "Controller Creation".

  For information about tuning traditional MPC controllers, see "Refinement".
- Reference and measured disturbance previewing are not supported. At each control interval, the current n_y reference and n_v measured disturbance signals apply for the entire prediction horizon.

- To limit the number of regions needed by explicit MPC, include only essential constraints.
  - When including a constraint on a manipulated variable (MV) use a short control horizon or MV blocking. See “Choosing Sample Time and Horizons”.
  - Avoid constraints on plant outputs. If such a constraint is essential, consider imposing it for selected prediction horizon steps rather than the entire prediction horizon.
- Establish upper and lower bounds for each of the $n_x = n_{xc} + n_y + n_v$ independent variables. You might know some of these bounds a priori. However, you must run simulations that record at least the n_{xc} controller states as the system operates over the range of expected conditions. It is very important that you not understimate this range, because the explicit MPC control function is not defined for independent variables outside the range.

  For information about specifying bounds, see `generateExplicitRange`.

  For information about simulating a traditional MPC controller, see “Simulation”.

Explicit MPC Generation

Given the constant MPC design parameters and the n_x upper and lower bounds on the control law’s independent variables, i.e.,

$$x_l \le x(k) \le x_u,$$

the `generateExplicitMPC` command determines n_r regions. Each of these regions is defined by an inequality constraint and the corresponding control law constants:

$$H_i x(k) \le K_i, \quad i = 1, n_r$$
$$u(k) = F_i x(k) + G_i, \quad i = 1, n_r.$$

The Explicit MPC Controller object contains the constants H_i, K_i, F_i, and G_i for each region. The Explicit MPC Controller object also holds the original (implicit) design and independent variable bounds. Provided that $x(k)$ stays within the specified bounds and you retain all n_r regions, the explicit MPC object should provide the same optimal MV adjustments, $u(k)$, as the equivalent implicit MPC object.

For details about explicit MPC, see [1]. For details about how the explicit MPC controller is generated, see [2].

Explicit MPC Simplification

Even a relatively simple explicit MPC controller might need $n_r >> 100$ to characterize the QP solution completely. If the number of regions is large, consider the following:

- Visualize the solution using the `plotSection` command.
- Use the `simplify` command to reduce the number of regions. In some cases, this can be done with no (or negligible) impact on control law optimality. For example, pairs of adjacent regions might employ essentially the same F_i and K_i constants. If so, and if the union of the two regions forms a convex set, they can be merged into a single region.

  Alternatively, you can eliminate relatively small regions or retain selected regions only. If during operation the current $x(k)$ is not contained in any of the retained regions, the explicit MPC will return a suboptimal $u(k)$, as follows:

  $$u(k) = F_j x(k) + G_j.$$

  Here, j is the index of the region whose bounding constraint, $H_j x(k) \le K_j$, is least violated.

Implementation

During operation, for a given $x(k)$, the explicit MPC controller performs the following steps:

1. Verifies that $x(k)$ satisfies the specified bounds, $x_l \le x(k) \le x_u$. If not, the controller returns an error status and sets $u(k) = u(k-1)$.
2. Beginning with region $i = 1$, tests the regions one by one to determine whether $x(k)$ belongs. If $H_i x(k) \le K_i$, then $x(k)$ belongs to region i. If $x(k)$ belongs to region i, then the controller:
   - Obtains F_i and G_i from memory, and computes $u(k) = F_i x(k) + G_i$.
   - Signals successful completion, by returning a status code and the index i.
   - Returns without testing the remaining regions.

If $x(k)$ does not belong to region i, the controller:

- Computes the violation term v_i, which is the largest (positive) component of the vector $(H_i x(k) - K_i)$.
- If v_i is the minimum violation for this $x(k)$, the controller sets $j = i$, and sets $v_{min} = v_i$.
- The controller then increments i and tests the next region.

3 If all regions have been tested and $x(k)$ does not belong to any region (for example, due to a numerical precision issue), the controller:

- Obtains F_j and G_j from memory, and computes $u(k) = F_j x(k) + G_j$.
- Sets status to indicate a suboptimal solution and returns.

Thus, the maximum computational time per control interval is that needed to test each region, computing the violation term in each case, and then calculating the suboptimal control adjustment.

Simulation

You can perform command-line simulations using the `sim` or `mpcmoveExplicit` commands.

You can use the Explicit MPC Controller block to connect an explicit MPC to a plant modeled in Simulink.

References

[1] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[2] A. Bemporad, "A multi-parametric quadratic programming algorithm with polyhedral computations based on nonnegative least squares," 2014, Submitted for publication.

See Also

Explicit MPC Controller | `generateExplicitMPC` | `mpcmoveExplicit`

Related Examples

- "Explicit MPC Control of a Single-Input-Single-Output Plant"
- "Explicit MPC Control of an Aircraft with Unstable Poles"
- "Explicit MPC Control of DC Servomotor with Constraint on Unmeasured Output"

More About

- "Explicit MPC" on page 6-2

Explicit MPC Control of a Single-Input-Single-Output Plant

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

See also MPCDOUBLEINT.

Define Plant Model

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

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

Design MPC Controller

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

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

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

Specify actuator saturation limits as MV constraints.

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

Generate Explicit MPC Controller

Explicit MPC executes the equivalent explicit piecewise affine version of the MPC control law defined by the traditional MPC. To generate an Explicit MPC from a traditional MPC, you must specify range for each controller state, reference signal, manipulated variable and measured disturbance so that the multi-parametric quadratic programming problem is solved in the parameter space defined by these ranges.

Obtain a range structure for initialization

Use `generateExplicitRange` command to obtain a range structure where you can specify range for each parameter afterwards.

```
range = generateExplicitRange(mpcobj);

-->Converting the "Model.Plant" property of "mpc" object to state-space.
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

Specify ranges for controller states

MPC controller states include states from plant model, disturbance model and noise model in that order. Setting the range of a state variable is sometimes difficult when the state does not correspond to a physical parameter. In that case, multiple runs of open-loop plant simulation with typical reference and disturbance signals are recommended in order to collect data that reflect the ranges of states.

```
range.State.Min(:) = [-10;-10];
range.State.Max(:) = [10;10];
```

Specify ranges for reference signals

Usually you know the practical range of the reference signals being used at the nominal operating point in the plant. The ranges used to generate Explicit MPC must be at least as large as the practical range.

```
range.Reference.Min = -2;
range.Reference.Max = 2;
```

Specify ranges for manipulated variables

If manipulated variables are constrained, the ranges used to generate Explicit MPC must be at least as large as these limits.

```
range.ManipulatedVariable.Min = -1.1;
range.ManipulatedVariable.Max = 1.1;
```

Construct the Explicit MPC controller

Use `generateExplicitMPC` command to obtain the Explicit MPC controller with the parameter ranges previously specified.

```
mpcobjExplicit = generateExplicitMPC(mpcobj, range);
display(mpcobjExplicit);
```

```
Regions found / unexplored:       19/       0


Explicit MPC Controller
-----------------------------------------------
Controller sample time:    0.1 (seconds)
Polyhedral regions:        19
Number of parameters:      4
Is solution simplified:    No
State Estimation:          Default Kalman gain
-----------------------------------------------
Type 'mpcobjExplicit.MPC' for the original implicit MPC design.
Type 'mpcobjExplicit.Range' for the valid range of parameters.
Type 'mpcobjExplicit.OptimizationOptions' for the options used in multi-parametirc QP c
Type 'mpcobjExplicit.PiecewiseAffineSolution' for regions and gain in each solution.
```

Use `simplify` command with the "exact" method to join pairs of regions whose corresponding gains are the same and whose union is a convex set. This practice can reduce memory footprint of the Explicit MPC controller without sacrifice any performance.

```
mpcobjExplicitSimplified = simplify(mpcobjExplicit, 'exact');
display(mpcobjExplicitSimplified);
```

```
Regions to analyze:       15/      15


Explicit MPC Controller
-----------------------------------------------
Controller sample time:    0.1 (seconds)
Polyhedral regions:        15
Number of parameters:      4
Is solution simplified:    Yes
State Estimation:          Default Kalman gain
-----------------------------------------------
Type 'mpcobjExplicitSimplified.MPC' for the original implicit MPC design.
Type 'mpcobjExplicitSimplified.Range' for the valid range of parameters.
Type 'mpcobjExplicitSimplified.OptimizationOptions' for the options used in multi-paran
Type 'mpcobjExplicitSimplified.PiecewiseAffineSolution' for regions and gain in each so
```

The number of piecewise affine region has been reduced.

Plot Piecewise Affine Partition

You can review any 2-D section of the piecewise affine partition defined by the Explicit MPC control law.

Obtain a plot parameter structure for initialization

Use `generatePlotParameters` command to obtain a parameter structure where you can specify which 2-D section to plot afterwards.

```
params = generatePlotParameters(mpcobjExplicitSimplified);
```

Specify parameters for a 2-D plot

In this example, you plot the 1th state variable vs. the 2nd state variable. All the other parameters must be fixed at a value within its range.

```
params.State.Index = [];
params.State.Value = [];
```

Fix other reference signals

```
params.Reference.Index = 1;
params.Reference.Value = 0;
```

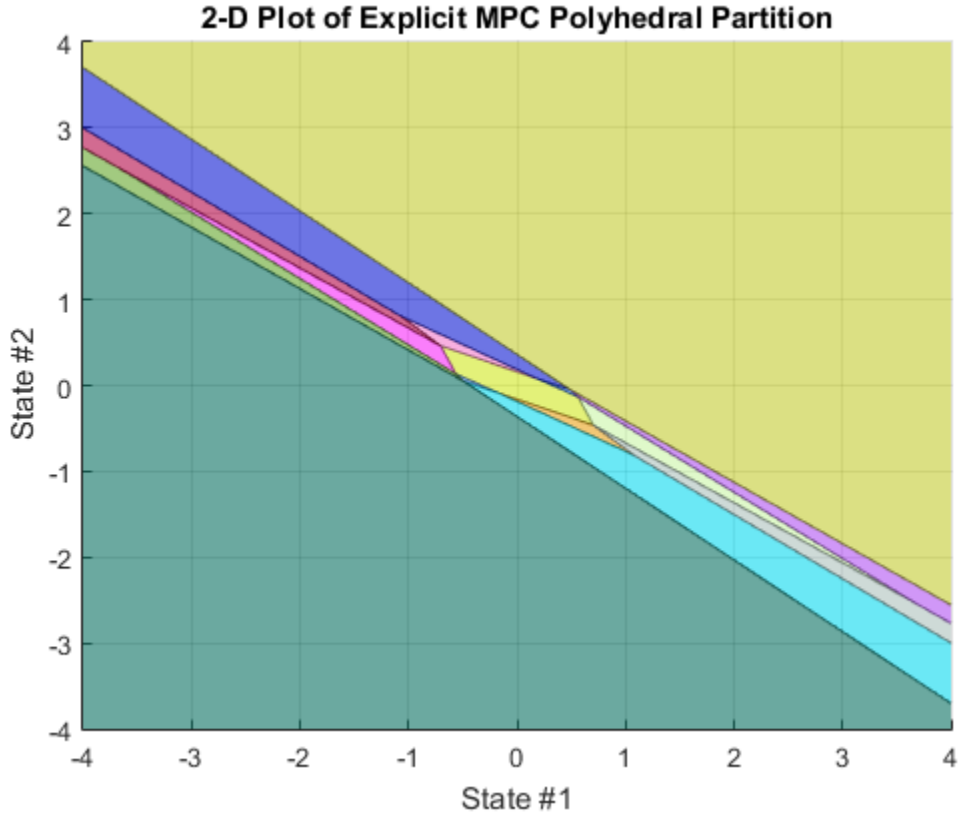
Fix manipulated variables

```
params.ManipulatedVariable.Index = 1;
params.ManipulatedVariable.Value = 0;
```

Plot the 2-D section

Use `plotSection` command to plot the 2-D section defined previously.

```
plotSection(mpcobjExplicitSimplified, params);
axis([-4 4 -4 4]);
grid
xlabel('State #1');
ylabel('State #2');
```



Simulate Using MPCMOVE Command

Compare closed-loop simulation between tradition MPC (as referred as Implicit MPC) and Explicit MPC using `mpcmove` and `mpcmoveExplicit` commands respectively.

Prepare to store the closed-loop MPC responses.

```
Tf = round(5/Ts);
YY = zeros(Tf,1);
YYExplicit = zeros(Tf,1);
UU = zeros(Tf,1);
UUExplicit = zeros(Tf,1);
```

Prepare the real plant used in simulation

```
sys = c2d(ss(plant),Ts);
xsys = [0;0];
xsysExplicit = xsys;
```

Use MPCSTATE object to specify the initial states for both controllers

```
xmpc = mpcstate(mpcobj);
xmpcExplicit = mpcstate(mpcobjExplicitSimplified);
```

Simulate closed-loop response in each iteration.

```
for t = 0:Tf
    % update plant measurement
    ysys = sys.C*xsys;
    ysysExplicit = sys.C*xsysExplicit;
    % compute traditional MPC action
    u = mpcmove(mpcobj,xmpc,ysys,1);
    % compute Explicit MPC action
    uExplicit = mpcmoveExplicit(mpcobjExplicit,xmpcExplicit,ysysExplicit,1);
    % store signals
    YY(t+1)=ysys;
    YYExplicit(t+1)=ysysExplicit;
    UU(t+1)=u;
    UUExplicit(t+1)=uExplicit;
    % update plant state
    xsys = sys.A*xsys + sys.B*u;
    xsysExplicit = sys.A*xsysExplicit + sys.B*uExplicit;
end
fprintf('\nDifference between traditional and Explicit MPC responses using MPCMOVE comm
```

```
Difference between traditional and Explicit MPC responses using MPCMOVE command is 1.80
```

Simulate Using SIM Command

Compare closed-loop simulation between tradition MPC and Explicit MPC using `sim` commands respectively.

```
Tf = 5/Ts;                        % simulation iterations
[y1,t1,u1] = sim(mpcobj,Tf,1);  % simulation with tradition MPC
[y2,t2,u2] = sim(mpcobjExplicitSimplified,Tf,1);   % simulation with Explicit MPC
```

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

```
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting the "Model.Plant" property of "mpc" object to state-space.
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting the "Model.Plant" property of "mpc" object to state-space.
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

The simulation results are identical.

```
fprintf('\nDifference between traditional and Explicit MPC responses using SIM command
```

```
Difference between traditional and Explicit MPC responses using SIM command is 1.80294e
```

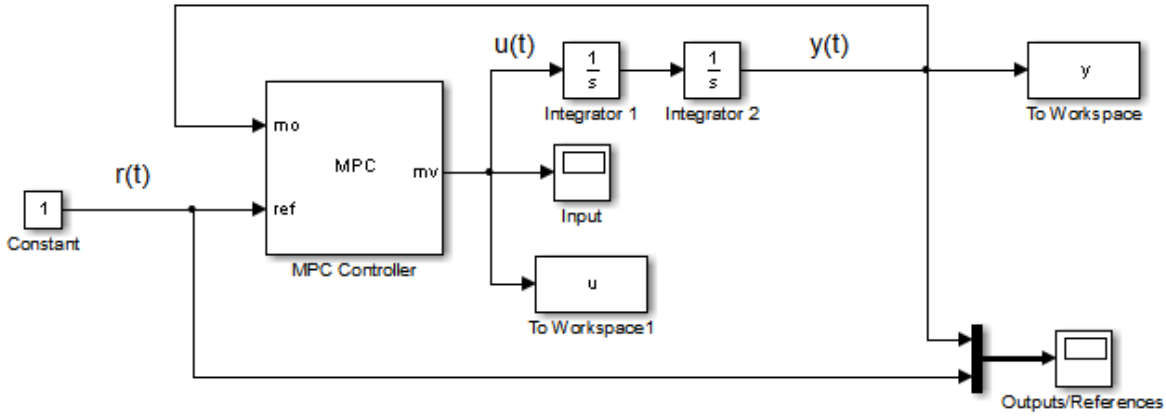
Simulate Using Simulink®

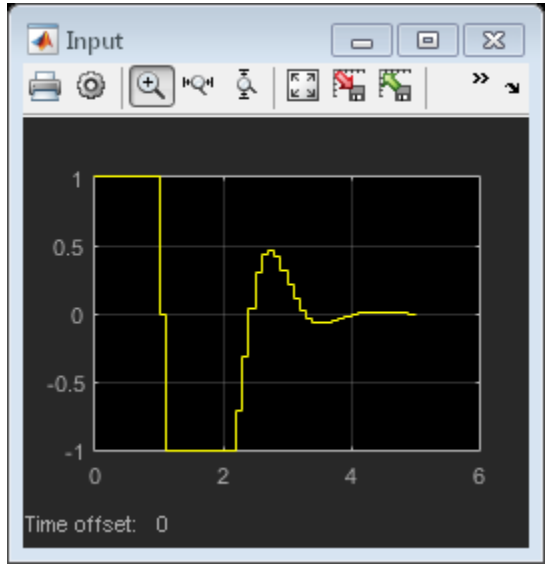
To run this example, Simulink® is required.

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

Simulate with traditional MPC controller in Simulink. Controller "mpcobj" is specified in the block dialog.

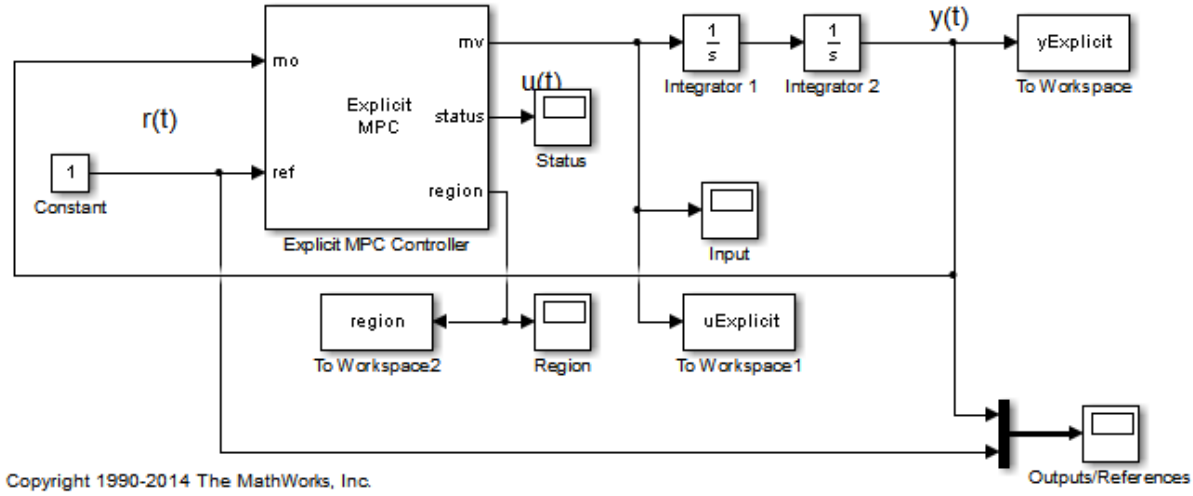
```
mdl = 'mpc_doubleint';
open_system(mdl);
sim(mdl);
```

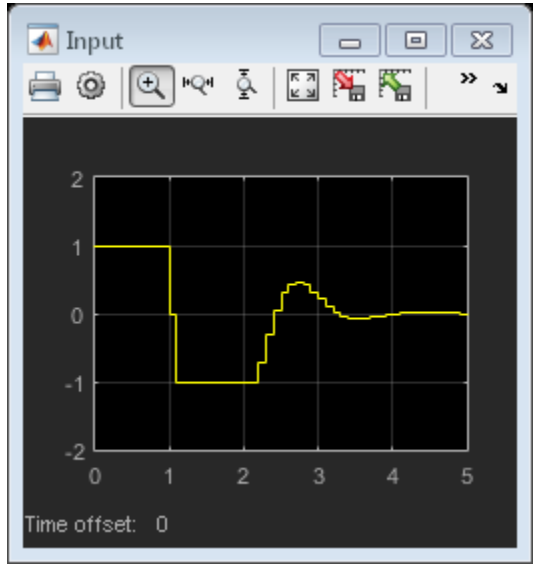


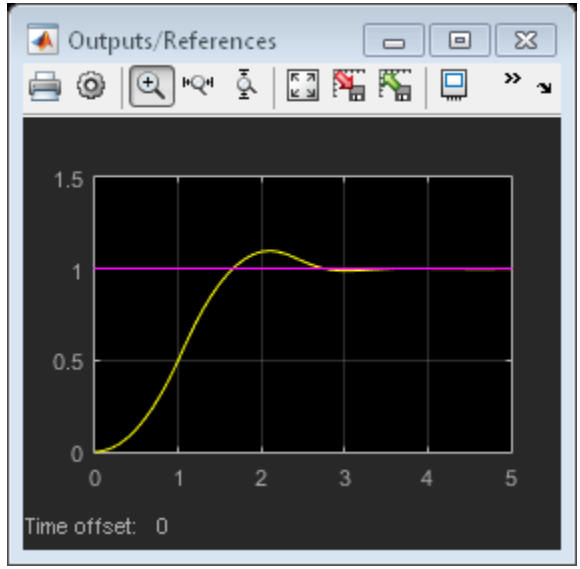


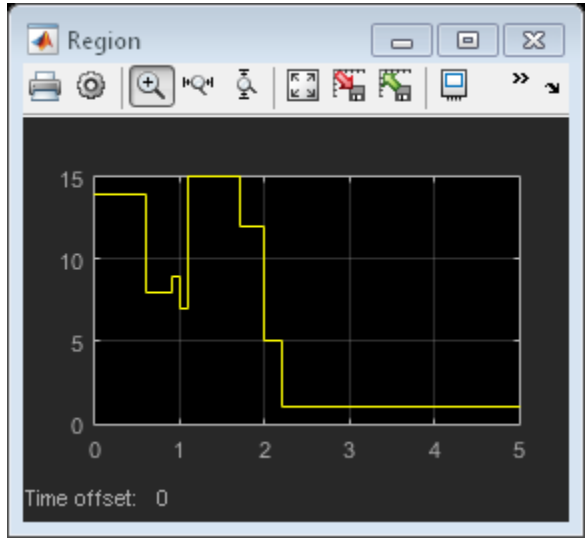
Simulate with Explicit MPC controller in Simulink. Controller "mpcobjExplicitSimplified" is specified in the block dialog.

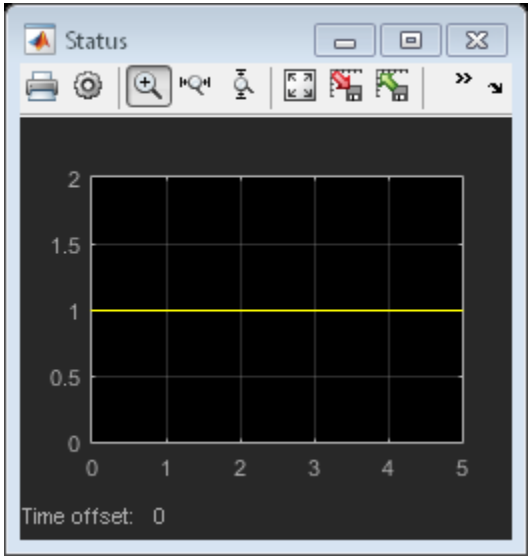
```
mdlExplicit = 'empc_doubleint';
open_system(mdlExplicit);
sim(mdlExplicit);
```


mo
ref
mv
status
region
Explicit MPC
Explicit MPC Controller
r(t)
1
Constant
u(t)
Status
1/s
Integrator 1
1/s
Integrator 2
y(t)
yExplicit
To Workspace
Input
region
To Workspace2
Region
uExplicit
To Workspace1
Outputs/References
Copyright 1990-2014 The MathWorks, Inc.


Input
2
1
0
-1
-2
0
1
2
3
4
5
Time offset: 0


Outputs/References
1.5
1
0.5
0
0
1
2
3
4
5
Time offset: 0


Region
15
10
5
0
0
1
2
3
4
5
Time offset: 0



The closed-loop responses are identical.

```
fprintf('\nDifference between traditional and Explicit MPC responses in Simulink is %g'
```

```
Difference between traditional and Explicit MPC responses in Simulink is 1.56789e-13
```

```
bdclose(mdl)
bdclose(mdlExplicit)
```

Related Examples

- “Explicit MPC Control of an Aircraft with Unstable Poles”
- “Explicit MPC Control of DC Servomotor with Constraint on Unmeasured Output”

More About

- “Explicit MPC” on page 6-2

Explicit MPC Control of an Aircraft with Unstable Poles

This example shows how to control an unstable aircraft with saturating actuators using Explicit MPC.

Reference:

[1] P. Kapasouris, M. Athans and G. Stein, "Design of feedback control systems for unstable plants with saturating actuators", Proc. IFAC Symp. on Nonlinear Control System Design, Pergamon Press, pp.302--307, 1990

[2] A. Bemporad, A. Casavola, and E. Mosca, "Nonlinear control of constrained linear systems via predictive reference management", IEEE® Trans. Automatic Control, vol. AC-42, no. 3, pp. 340-349, 1997.

See also MPCAIRCRAFT.

Define Aircraft Model

The linear open-loop dynamic model is as follows:

```
A = [-0.0151 -60.5651 0 -32.174;
     -0.0001 -1.3411 0.9929 0;
      0.00018 43.2541 -0.86939 0;
      0      0       1      0];
B = [-2.516 -13.136;
     -0.1689 -0.2514;
     -17.251 -1.5766;
      0        0];
C = [0 1 0 0;
     0 0 0 1];
D = [0 0;
     0 0];
plant = ss(A,B,C,D);
x0 = zeros(4,1);
```

The manipulated variables are the elevator and flaperon angles, the attack and pitch angles are measured outputs to be regulated.

The open-loop response of the system is unstable.

```
pole(plant)
```

```
ans =

  -7.6636 + 0.0000i
   5.4530 + 0.0000i
  -0.0075 + 0.0556i
  -0.0075 - 0.0556i
```

Design MPC Controller

To obtain an Explicit MPC controller, you must first design a traditional MPC (also referred as Implicit MPC) that is able to achieves your control objectives.

```
% *MV Constraints*
```

Both manipulated variables are constrained between +/- 25 degrees. Since the plant inputs and outputs are of different orders of magnitude, you also use scale factors to faciliate MPC tuning. Typical choices of scale factor are the upper/lower limit or the operating range.

```
MV = struct('Min',{-25,-25},'Max',{25,25},'ScaleFactor',{50,50});
```

OV Constraints

Both plant outputs have constraints to limit undershoots at the first prediction horizon. You also specify scale factors for outputs.

```
OV = struct('Min',{[-0.5;-Inf],[-100;-Inf]},'Max',{[0.5;Inf],[100;Inf]},'ScaleFactor',
```

Weights

The control task is to get zero offset for piecewise-constant references, while avoiding instability due to input saturation. Because both MV and OV variables are already scaled in MPC controller, MPC weights are dimensionless and applied to the scaled MV and OV values. In this example, you penalize the two outputs equally with the same OV weights.

```
Weights = struct('MV',[0 0],'MVRate',[0.1 0.1],'OV',[10 10]);
```

Construct the traditional MPC controller

Create an MPC controller with plant model, sample time and horizons.

```
Ts = 0.05;          % Sampling time
p = 10;             % Prediction horizon
m = 2;              % Control horizon
mpcobj = mpc(plant,Ts,p,m,Weights,MV,OV);
```

Generate Explicit MPC Controller

Explicit MPC executes the equivalent explicit piecewise affine version of the MPC control law defined by the traditional MPC. To generate an Explicit MPC from a traditional MPC, you must specify range for each controller state, reference signal, manipulated variable and measured disturbance so that the multi-parametric quadratic programming problem is solved in the parameter space defined by these ranges.

Obtain a range structure for initialization

Use `generateExplicitRange` command to obtain a range structure where you can specify range for each parameter afterwards.

```
range = generateExplicitRange(mpcobj);
```

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

Specify ranges for controller states

MPC controller states include states from plant model, disturbance model and noise model in that order. Setting the range of a state variable is sometimes difficult when the state does not correspond to a physical parameter. In that case, multiple runs of open-loop plant simulation with typical reference and disturbance signals are recommended in order to collect data that reflect the ranges of states.

```
range.State.Min(:) = -10000;
range.State.Max(:) = 10000;
```

Specify ranges for reference signals

Usually you know the practical range of the reference signals being used at the nominal operating point in the plant. The ranges used to generate Explicit MPC must be at least as large as the practical range.

```
range.Reference.Min = [-1;-11];
range.Reference.Max = [1;11];
```

Specify ranges for manipulated variables

If manipulated variables are constrained, the ranges used to generate Explicit MPC must be at least as large as these limits.

```
range.ManipulatedVariable.Min = [MV(1).Min; MV(2).Min] - 1;
range.ManipulatedVariable.Max = [MV(1).Max; MV(2).Max] + 1;
```

Construct the Explicit MPC controller

Use `generateExplicitMPC` command to obtain the Explicit MPC controller with the parameter ranges previously specified.

```
mpcobjExplicit = generateExplicitMPC(mpcobj, range);
display(mpcobjExplicit);
```

```
Regions found / unexplored:      483/       0


Explicit MPC Controller
-----------------------------------------------
Controller sample time:    0.05 (seconds)
Polyhedral regions:        483
Number of parameters:      10
Is solution simplified:    No
State Estimation:          Default Kalman gain
-----------------------------------------------
Type 'mpcobjExplicit.MPC' for the original implicit MPC design.
Type 'mpcobjExplicit.Range' for the valid range of parameters.
Type 'mpcobjExplicit.OptimizationOptions' for the options used in multi-parametirc QP c
Type 'mpcobjExplicit.PiecewiseAffineSolution' for regions and gain in each solution.
```

Use `simplify` command with the "exact" method to join pairs of regions whose corresponding gains are the same and whose union is a convex set. This practice can reduce memory footprint of the Explicit MPC controller without sacrifice any performance.

```
mpcobjExplicitSimplified = simplify(mpcobjExplicit, 'exact');
display(mpcobjExplicitSimplified);
```

```
Regions to analyze:      471/     471


Explicit MPC Controller
-----------------------------------------------
Controller sample time:    0.05 (seconds)
```

```
Polyhedral regions:        471
Number of parameters:      10
Is solution simplified:    Yes
State Estimation:          Default Kalman gain
-----------------------------------------
Type 'mpcobjExplicitSimplified.MPC' for the original implicit MPC design.
Type 'mpcobjExplicitSimplified.Range' for the valid range of parameters.
Type 'mpcobjExplicitSimplified.OptimizationOptions' for the options used in multi-para
Type 'mpcobjExplicitSimplified.PiecewiseAffineSolution' for regions and gain in each s
```

The number of piecewise affine region has been reduced.

Plot Piecewise Affine Partition

You can review any 2-D section of the piecewise affine partition defined by the Explicit MPC control law.

Obtain a plot parameter structure for initialization

Use `generatePlotParameters` command to obtain a parameter structure where you can specify which 2-D section to plot afterwards.

```
params = generatePlotParameters(mpcobjExplicitSimplified);
```

Specify parameters for a 2-D plot

In this example, you plot the pitch angle (the 4th state variable) vs. its reference (the 2nd reference signal). All the other parameters must be fixed at a value within its range.

Fix other state variables

```
params.State.Index = [1 2 3 5 6];
params.State.Value = [0 0 0 0 0];
```

Fix other reference signals

```
params.Reference.Index = 1;
params.Reference.Value = 0;
```

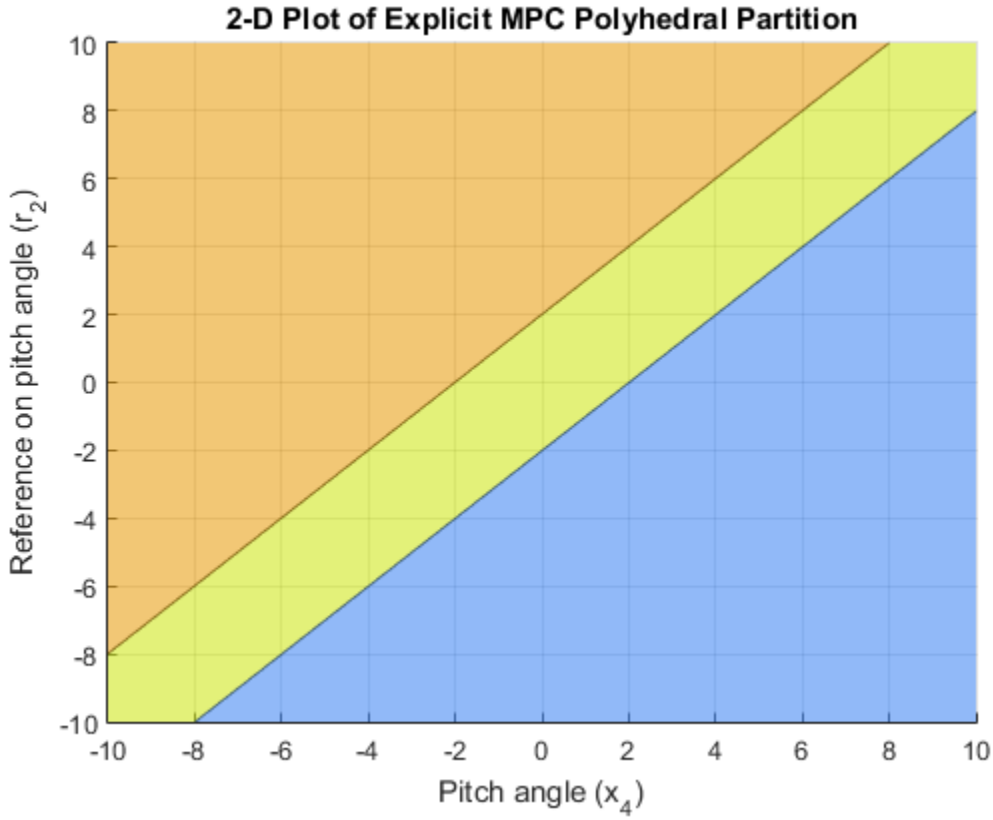
Fix manipulated variables

```
params.ManipulatedVariable.Index = [1 2];
params.ManipulatedVariable.Value = [0 0];
```

Plot the 2-D section

Use `plotSection` command to plot the 2-D section defined previously.

```
plotSection(mpcobjExplicitSimplified, params);
axis([-10 10 -10 10]);
grid;
xlabel('Pitch angle (x_4)');
ylabel('Reference on pitch angle (r_2)');
```



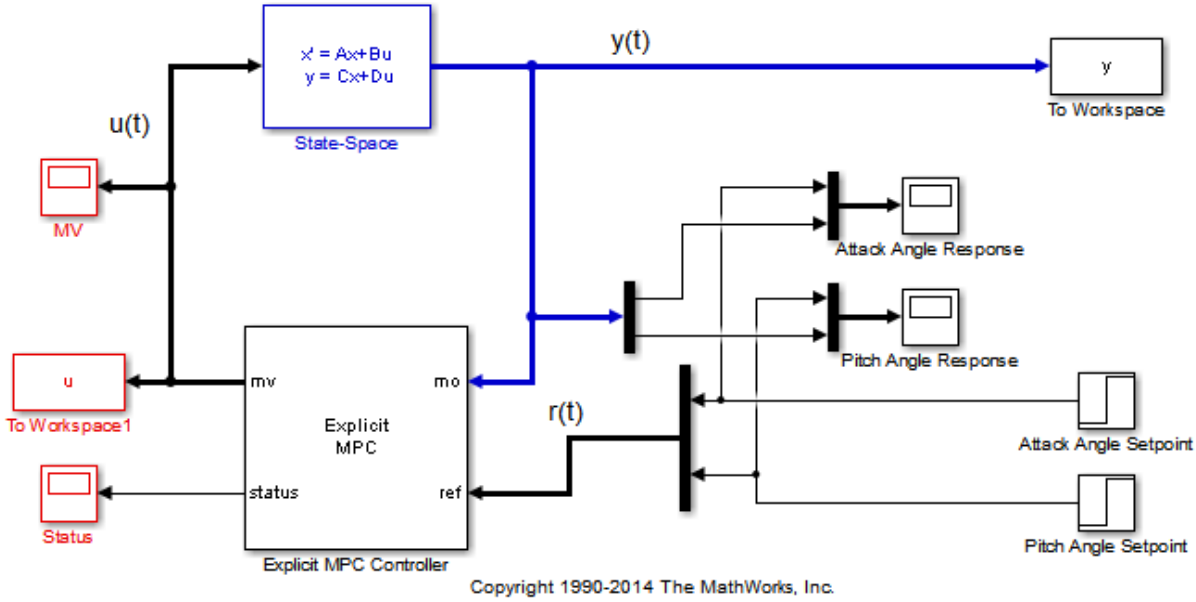
Simulate Using Simulink®

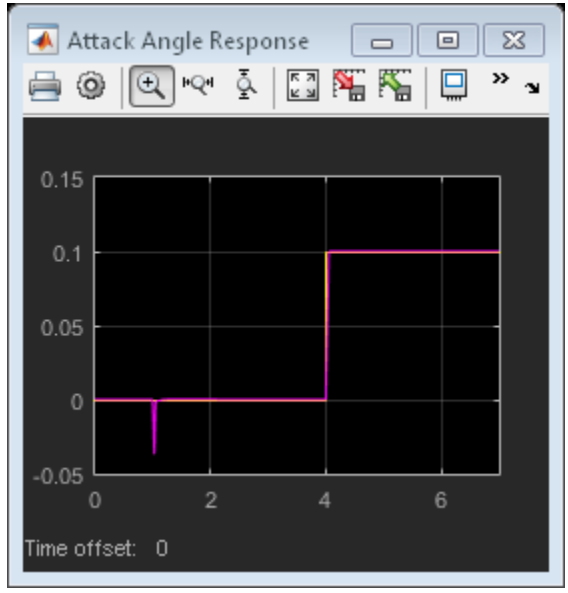
To run this example, Simulink® is required.

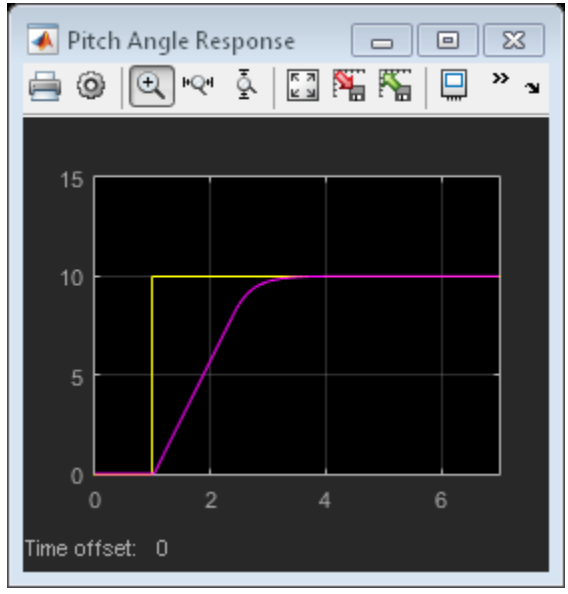
```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
```

Simulate closed-loop control of the linear plant model in Simulink, using the Explicit MPC Controller block. Controller "mpcobjExplicitSimplified" is specified in the block dialog.

```
mdl = 'empc_aircraft';
open_system(mdl)
sim(mdl)
```




Attack Angle Response
0.15
0.1
0.05
0
-0.05
0
2
4
6
Time offset: 0


Pitch Angle Response
15
10
5
0
0
2
4
6
Time offset: 0

The closed-loop response is identical to the traditional MPC controller designed in the "mpcaircraft" example.

```
bdclose(mdl)
```

Related Examples

- “Explicit MPC Control of a Single-Input-Single-Output Plant”
- “Explicit MPC Control of DC Servomotor with Constraint on Unmeasured Output”

More About

- “Explicit MPC” on page 6-2

Explicit MPC Control of DC Servomotor with Constraint on Unmeasured Output

This example shows how to use Explicit MPC to control DC servomechanism under voltage and shaft torque constraints.

Reference

[1] A. Bemporad and E. Mosca, "Fulfilling hard constraints in uncertain linear systems by reference managing," Automatica, vol. 34, no. 4, pp. 451-461, 1998.

See also MPCMOTOR.

Define DC-Servo Motor Model

The linear open-loop dynamic model is defined in "plant". Variable "tau" is the maximum admissible torque to be used as an output constraint.

```
[plant, tau] = mpcmotormodel;
```

Design MPC Controller

Specify input and output signal types for the MPC controller. The second output, torque, is unmeasurable.

```
plant = setmpcsignals(plant,'MV',1,'MO',1,'UO',2);
```

MV Constraints

The manipulated variable is constrained between +/- 220 volts. Since the plant inputs and outputs are of different orders of magnitude, you also use scale factors to faciliate MPC tuning. Typical choices of scale factor are the upper/lower limit or the operating range.

```
MV = struct('Min',-220,'Max',220,'ScaleFactor',440);
```

OV Constraints

Torque constraints are only imposed during the first three prediction steps to limit the complexity of the explicit MPC design.

```
OV = struct('Min',{Inf, [-tau;-tau;-tau;-Inf]},'Max',{Inf, [tau;tau;tau;Inf]},'ScaleFac
```

Weights

The control task is to get zero tracking offset for the angular position. Since you only have one manipulated variable, the shaft torque is allowed to float within its constraint by setting its weight to zero.

```
Weights = struct('MV',0,'MVRate',0.1,'OV',[0.1 0]);
```

Construct MPC controller

Create an MPC controller with plant model, sample time and horizons.

```
Ts = 0.1;           % Sampling time
p = 10;             % Prediction horizon
m = 2;              % Control horizon
mpcobj = mpc(plant,Ts,p,m,Weights,MV,OV);
```

Generate Explicit MPC Controller

Explicit MPC executes the equivalent explicit piecewise affine version of the MPC control law defined by the traditional MPC. To generate an Explicit MPC from a traditional MPC, you must specify the range for each controller state, reference signal, manipulated variable and measured disturbance so that the multi-parametric quadratic programming problem is solved in the parameter sets defined by these ranges.

Obtain a range structure for initialization

Use `generateExplicitRange` command to obtain a range structure where you can specify the range for each parameter afterwards.

```
range = generateExplicitRange(mpcobj);
```

```
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

Specify ranges for controller states

MPC controller states include states from plant model, disturbance model and noise model in that order. Setting the range of a state variable is sometimes difficult when the

state does not correspond to a physical parameter. In that case, multiple runs of open-loop plant simulation with typical reference and disturbance signals are recommended in order to collect data that reflect the ranges of states.

```
range.State.Min(:) = -1000;
range.State.Max(:) = 1000;
```

Specify ranges for reference signals

Usually you know the practical range of the reference signals being used at the nominal operating point in the plant. The ranges used to generate Explicit MPC must be at least as large as the practical range. Note that the range for torque reference is fixed at 0 because it has zero weight.

```
range.Reference.Min = [-5;0];
range.Reference.Max = [5;0];
```

Specify ranges for manipulated variables

If manipulated variables are constrained, the ranges used to generate Explicit MPC must be at least as large as these limits.

```
range.ManipulatedVariable.Min = MV.Min - 1;
range.ManipulatedVariable.Max = MV.Max + 1;
```

Construct the Explicit MPC controller

Use `generateExplicitMPC` command to obtain the Explicit MPC controller with the parameter ranges previously specified.

```
mpcobjExplicit = generateExplicitMPC(mpcobj, range);
display(mpcobjExplicit);
```

```
Regions found / unexplored:       75/       0


Explicit MPC Controller
-----------------------------------------
Controller sample time:    0.1 (seconds)
Polyhedral regions:        75
Number of parameters:      6
```

```
Is solution simplified:    No
State Estimation:          Default Kalman gain
-------------------------------------------
Type 'mpcobjExplicit.MPC' for the original implicit MPC design.
Type 'mpcobjExplicit.Range' for the valid range of parameters.
Type 'mpcobjExplicit.OptimizationOptions' for the options used in multi-parametirc QP c
Type 'mpcobjExplicit.PiecewiseAffineSolution' for regions and gain in each solution.
```

Plot Piecewise Affine Partition

You can review any 2-D section of the piecewise affine partition defined by the Explicit MPC control law.

Obtain a plot parameter structure for initialization

Use `generatePlotParameters` command to obtain a parameter structure where you can specify which 2-D section to plot afterwards.

```
params = generatePlotParameters(mpcobjExplicit);
```

Specify parameters for a 2-D plot

In this example, you plot the 1th state variable vs. the 2nd state variable. All the other parameters must be fixed at a value within its range.

Fix other state variables

```
params.State.Index = [3 4];
params.State.Value = [0 0];
```

Fix reference signals

```
params.Reference.Index = [1 2];
params.Reference.Value = [pi 0];
```

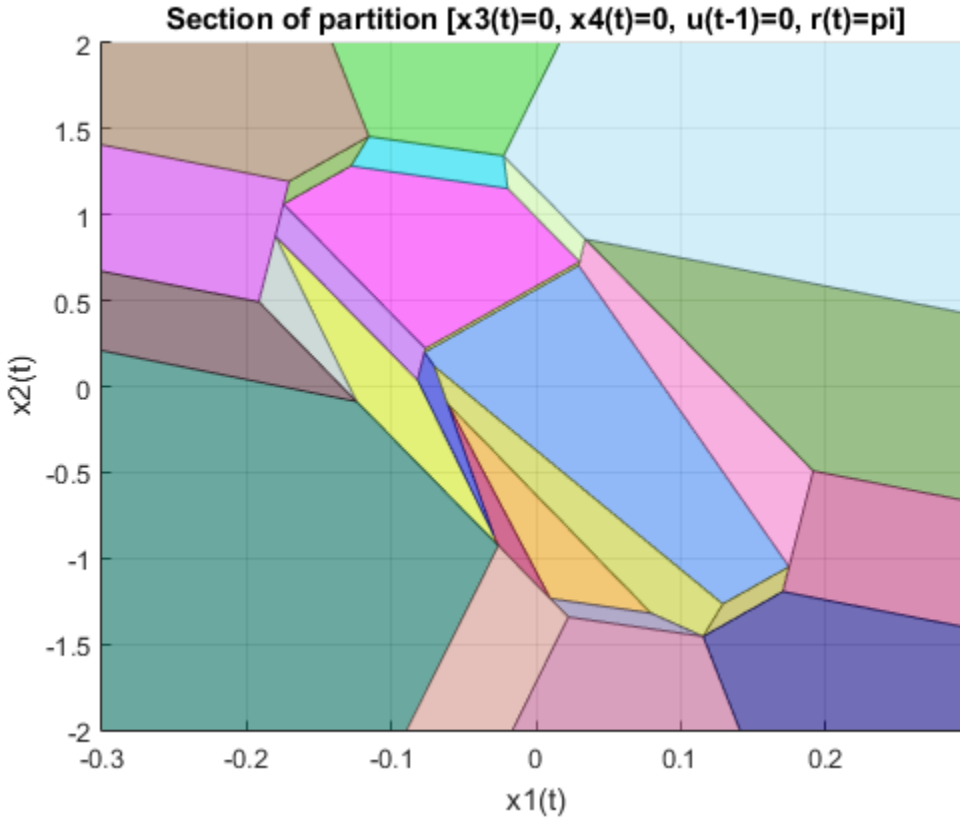
Fix manipulated variables

```
params.ManipulatedVariable.Index = 1;
params.ManipulatedVariable.Value = 0;
```

Plot the 2-D section

Use `plotSection` command to plot the 2-D section defined previously.

```
plotSection(mpcobjExplicit, params);
axis([-.3 .3 -2 2]);
grid
title('Section of partition [x3(t)=0, x4(t)=0, u(t-1)=0, r(t)=pi]')
xlabel('x1(t)');
ylabel('x2(t)');
```



Simulate Using SIM Command

Compare closed-loop simulation between traditional MPC (as referred as Implicit MPC) and Explicit MPC

```
Tstop = 8;                          % seconds
```

```
Tf = round(Tstop/Ts);           % simulation iterations
r = [pi 0];                     % reference signal
[y1,t1,u1] = sim(mpcobj,Tf,r);  % simulation with traditional MPC
[y2,t2,u2] = sim(mpcobjExplicit,Tf,r);   % simulation with Explicit MPC
```

```
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

The simulation results are identical.

```
fprintf('SIM command: Difference between QP-based and Explicit MPC trajectories = %g\n
```

```
SIM command: Difference between QP-based and Explicit MPC trajectories = 6.88112e-12
```

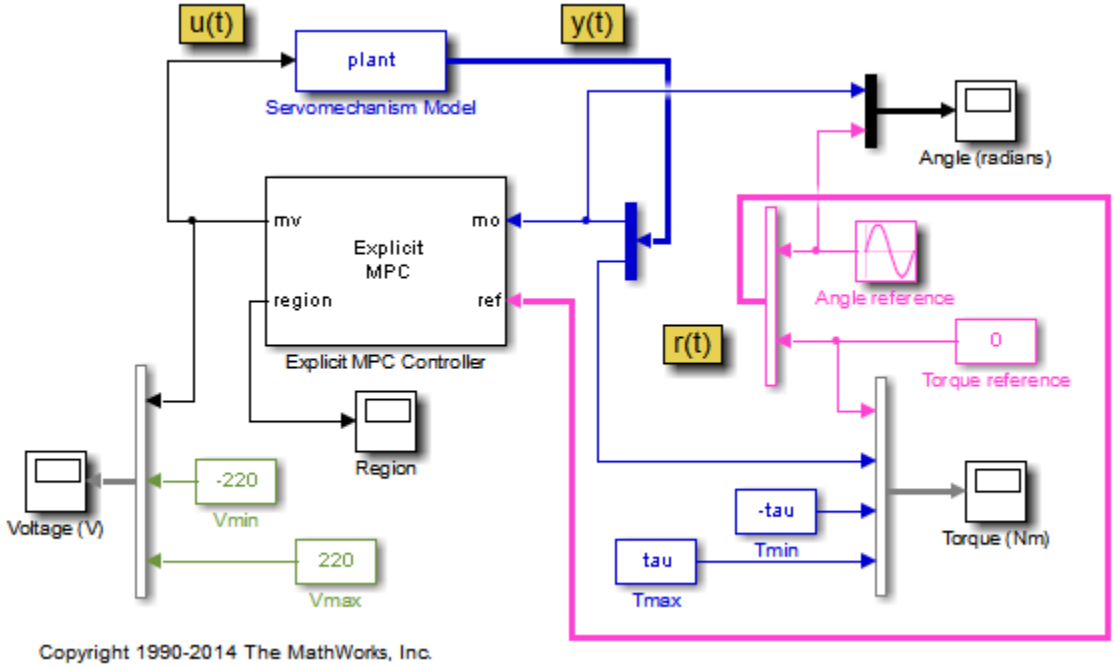
Simulate Using Simulink®

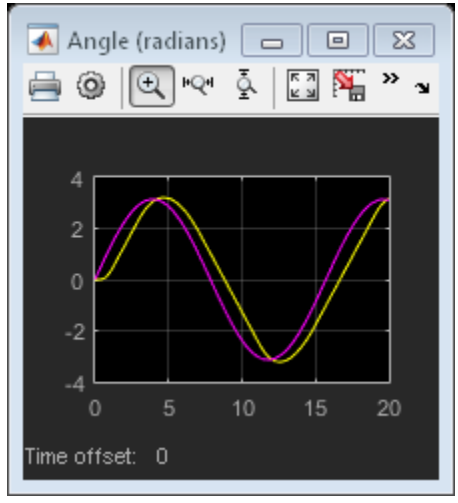
To run this example, Simulink® is required.

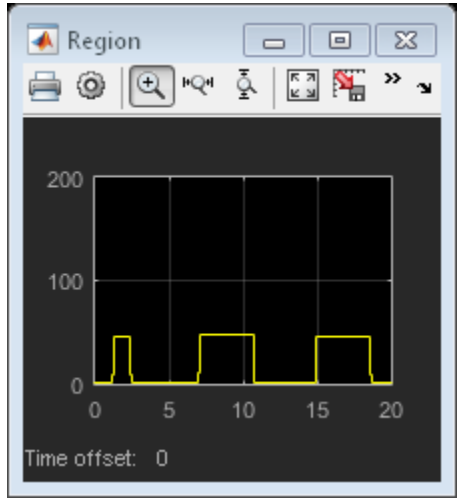
```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink(R) is required to run this example.')
    return
end
```

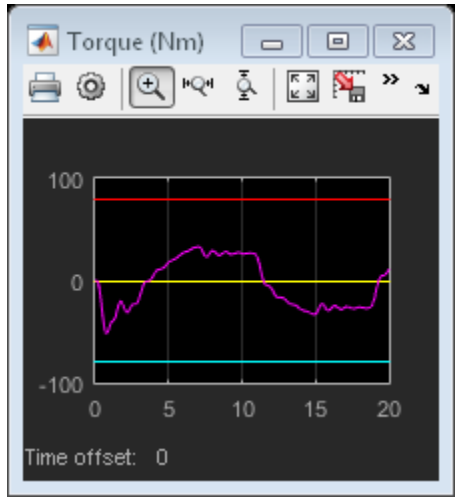
Simulate closed-loop control of the linear plant model in Simulink, using the Explicit MPC Controller block. Controller "mpcobjExplicit" is specified in the block dialog.

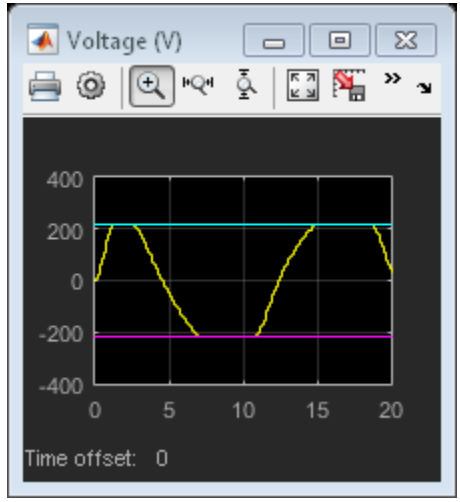
```
mdl = 'empc_motor';
open_system(mdl)
sim(mdl);
```


u(t)
plant
Servomechanism Model
y(t)
Angle (radians)
mv
mo
Explicit
MPC
region
ref
Explicit MPC Controller
Angle reference
r(t)
0
Torque reference
Region
Voltage (V)
-220
Vmin
220
Vmax
-tau
Tmin
tau
Tmax
Torque (Nm)
Copyright 1990-2014 The MathWorks, Inc.


Angle (radians)
4
2
0
-2
-4
0
5
10
15
20
Time offset: 0


Region
200
100
0
0
5
10
15
20
Time offset: 0


Torque (Nm)
100
0
-100
0
5
10
15
20
Time offset: 0



The closed-loop response is identical to the traditional MPC controller designed in the "mpcmotor" example.

Control Using Sub-optimal Explicit MPC

To reduce the memory footprint, you can use `simplify` command to reduce the number of piecewise affine solution regions. For example, you can remove regions whose Chebychev radius is smaller than .08. However, the price you pay is that the controler performance now becomes sub-optimal.

Use `simplify` command to generate Explicit MPC with sub-optimal solutions.

```
mpcobjExplicitSimplified = simplify(mpcobjExplicit, 'radius', 0.08);
disp(mpcobjExplicitSimplified);
```

```
Regions to analyze:       75/      75 --> 37 regions deleted.

  explicitMPC with properties:

                       MPC: [1x1 mpc]
                     Range: [1x1 struct]
       OptimizationOptions: [1x1 struct]
    PiecewiseAffineSolution: [1x38 struct]
```

```
                IsSimplified: 1
```

The number of piecewise affine regions has been reduced.

Compare closed-loop simulation between sub-optimal Explicit MPC and Explicit MPC.

```
[y3,t3,u3] = sim(mpcobjExplicitSimplified, Tf, r);
```

```
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

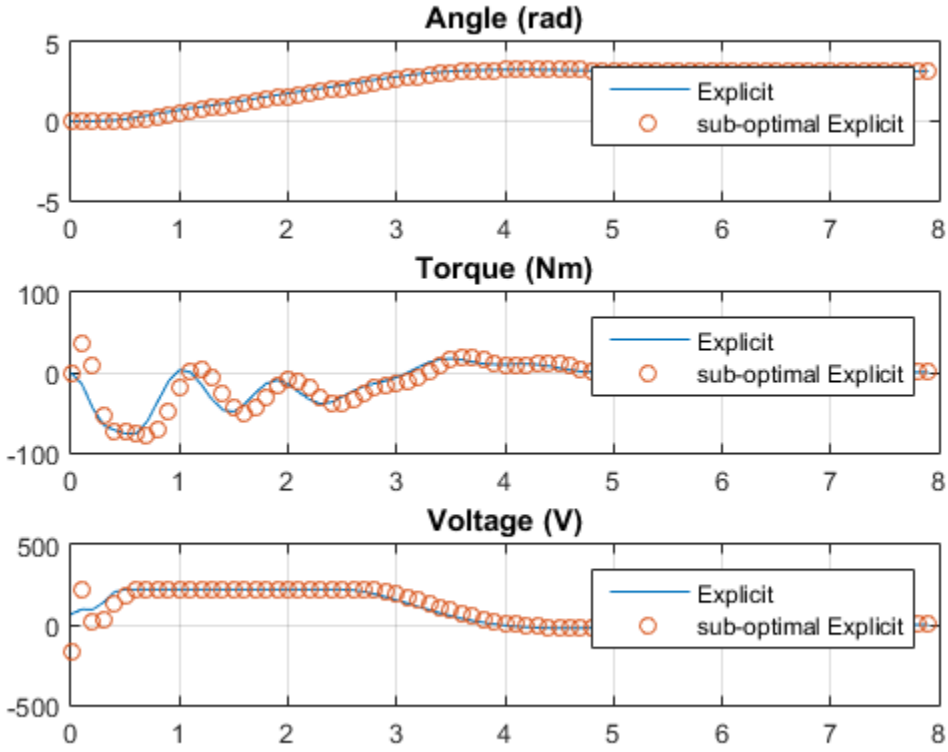
The simulation results are not the same.

```
fprintf('SIM command: Difference between exact and suboptimal MPC trajectories = %g\n'
```

```
SIM command: Difference between exact and suboptimal MPC trajectories = 439.399
```

Plot results.

```
figure;
subplot(3,1,1)
plot(t1,y1(:,1),t3,y3(:,1),'o');
grid
title('Angle (rad)')
legend('Explicit','sub-optimal Explicit')
subplot(3,1,2)
plot(t1,y1(:,2),t3,y3(:,2),'o');
grid
title('Torque (Nm)')
legend('Explicit','sub-optimal Explicit')
subplot(3,1,3)
plot(t1,u1,t3,u3,'o');
grid
title('Voltage (V)')
legend('Explicit','sub-optimal Explicit')
```



The simulation result with the sub-optimal Explicit MPC is slightly worse.

```
bdclose(mdl)
```

Related Examples

- "Explicit MPC Control of a Single-Input-Single-Output Plant"
- "Explicit MPC Control of an Aircraft with Unstable Poles"

More About

- "Explicit MPC" on page 6-2

Gain Scheduling MPC Design

Gain-Scheduled MPC

The Multiple MPC Controllers block for Simulink allows you to switch between a defined set of MPC Controllers. You might need this feature if the plant operating characteristics change in a predictable way, and the change is such that a single prediction model cannot provide adequate accuracy. This approach is comparable to the use of gain scheduling in conventional feedback control.

The individual MPC controllers coordinate to make switching from one to another bumpless, avoiding a sudden change in the manipulated variables when the switch occurs.

You can perform command-line simulations using the `mpcmoveMultiple` command.

More About

- "Design Workflow for Gain Scheduling"
- "Relationship of Multiple MPC Controllers to MPC Controller Block"

Design Workflow for Gain Scheduling

In this section...

General Design Steps

- Define and tune a nominal MPC controller for the most likely (or average) operating conditions. (See "MPC Design".)
- Use simulations to determine an operating condition at which the nominal controller loses robustness. See "Simulation".
- Identify a measurement (or combination of measurements) signaling when the nominal controller should be replaced.
- Determine a plant prediction model to be used at the new condition. Its input and output variables must be the same as in the nominal case.
- Define a new MPC controller based on the new prediction model. Use the nominal controller settings as a starting point, and test and retune controller settings if necessary.
- If two controllers are inadequate to provide robustness over the full operational range, consider adding another. If it appears that you need more than three controllers to provide robustness over the full range, consider using adaptive MPC instead. See "Adaptive MPC Design".
- In your Simulink model, configure the Multiple MPC Controllers block. Specify the set of MPC controllers to be used, and specify the switching criterion.
- Test in closed-loop simulation over the full operating range to verify robustness and bumpless switching.

Tips

- Recommended MPC start-up practice is a warm-up period in which the plant operates under manual control while the controller initializes its state estimate. This typically requires 10-20 control intervals. A warm-up is especially important for the Multiple MPC Controllers block. Otherwise, switching between MPC controllers might upset the manipulated variables.

- If you select the Multiple MPC Controllers block's custom state estimation option, all MPC controllers in the set must have the same state dimension. This places implicit restrictions on plant and disturbance models.

See Also

`mpcmoveMultiple` | Multiple MPC Controllers

Related Examples

- "Schedule Controllers at Multiple Operating Points"
- "Coordinate Multiple Controllers at Different Operating Points"
- "Gain Scheduled MPC Control of Nonlinear Chemical Reactor"
- "Gain Scheduled MPC Control of Mass-Spring System"

More About

- "Relationship of Multiple MPC Controllers to MPC Controller Block"

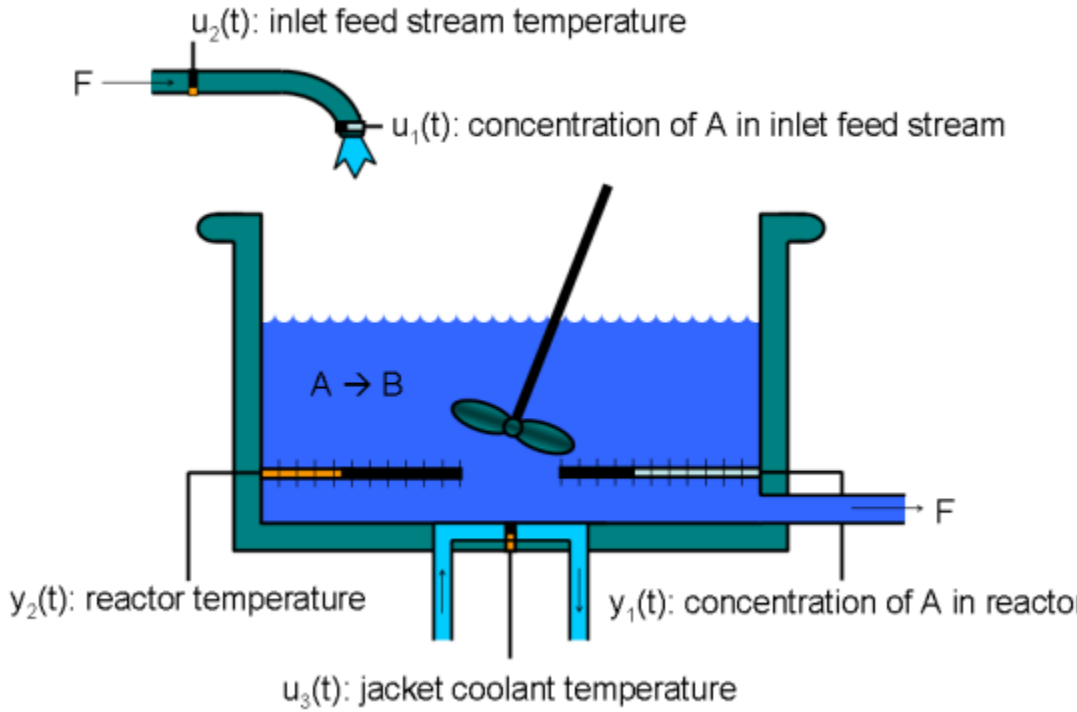
Gain Scheduled MPC Control of Nonlinear Chemical Reactor

This example shows how to use multiple MPC controllers to control a nonlinear continuous stirred tank reactor (CSTR) as it transitions from low conversion rate to high conversion rate.

Multiple MPC Controllers are designed at different operating conditions and then implemented with the Multiple MPC Controller block in Simulink. At run time, a scheduling signal is used to switch controller from one to another.

About the Continuous Stirred Tank Reactor

A Continuously Stirred Tank Reactor (CSTR) is a common chemical system in the process industry. A schematic of the CSTR system is:



This is a jacketed non-adiabatic tank reactor described extensively in Seborg's book, "Process Dynamics and Control", published by Wiley, 2004. The vessel is assumed to be

perfectly mixed, and a single first-order exothermic and irreversible reaction, A --> B, takes place. The inlet stream of reagent A is fed to the tank at a constant volumetric rate. The product stream exits continuously at the same volumetric rate and liquid density is constant. Thus the volume of reacting liquid is constant.

The inputs of the CSTR model are:

$u_1 = CA_i$ Concentration of A in inlet feed stream$[kgmol/m^3]$
$u_2 = T_i$ Inlet feed stream temperature$[K]$
$u_3 = T_c$ Jacket coolant temperature$[K]$

and the outputs (y(t)), which are also the states of the model (x(t)), are:

$y_1 = x_1 = CA$ Concentration of A in reactor tank$[kgmol/m^3]$
$y_2 = x_2 = T$ Reactor temperature$[K]$

The control objective is to maintain the concentration of reagent A, CA at its desired setpoint, which changes over time when reactor transitions from low conversion rate to high conversion rate. The coolant temperature T_c is the manipulated variable used by the MPC controller to track the reference. The inlet feed stream concentration and temperature are assumed to be constant. The Simulink model `mpc_cstr_plant` implements the nonlinear CSTR plant.

About Gain Scheduled Model Predictive Control

It is well known that the CSTR dynamics are strongly nonlinear with respect to reactor temperature variations and can be open-loop unstable during the transition from one operating condition to another. A single MPC controller designed at a particular operating condition cannot give satisfactory control performance over a wide operating range.

To control the nonlinear CSTR plant with linear MPC control technique, you have a few options:

- If a linear plant model cannot be obtained at run time, first you need to obtain several linear plant models offline at different operating conditions that cover the typical operating range. Next you can choose one of the two approaches to implement MPC control strategy:

(1) Design several MPC controllers offline, one for each plant model. At run time, use Multiple MPC Controller block that switches MPC controllers from one to another based on a desired scheduling strategy, as discussed in this example. Use this approach when the plant models have different orders or time delays.

(2) Design one MPC controller offline at a nominal operating point. At run time, use Adaptive MPC Controller block (updating predictive model at each control interval) together with Linear Parameter Varying (LPV) System block (supplying linear plant model with a scheduling strategy). See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Linear Parameter Varying System" for more details. Use this approach when all the plant models have the same order and time delay.

- If a linear plant model can be obtained at run time, you should use Adaptive MPC Controller block to achieve nonlinear control. There are two typical ways to obtain a linear plant model online:

(1) Use successive linearization. See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization" for more details. Use this approach when a nonlinear plant model is available and can be linearized at run time.

(2) Use online estimation to identify a linear model when loop is closed. See "Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation" for more details. Use this approach when linear plant model cannot be obtained from either an LPV system or successive linearization.

Obtain Linear Plant Model at Initial Operating Condition

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

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

First, a linear plant model is obtained at the initial operating condition, CAi is 10 kgmol/m^3, Ti and Tc are 298.15 K. Functions from Simulink Control Design such as "operspec", "findop", "linearize", are used to generate the linear state space system from the Simulink model.

Create operating point specification.

```
plant_mdl = 'mpc_cstr_plant';
op = operspec(plant_mdl);
```

Feed concentration is known at the initial condition.

```
op.Inputs(1).u = 10;
op.Inputs(1).Known = true;
```

Feed temperature is known at the initial condition.

```
op.Inputs(2).u = 298.15;
op.Inputs(2).Known = true;
```

Coolant temperature is known at the initial condition.

```
op.Inputs(3).u = 298.15;
op.Inputs(3).Known = true;
```

Compute initial condition.

```
[op_point, op_report] = findop(plant_mdl,op);
% Obtain nominal values of x, y and u.
x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];
```

```
 Operating Point Search Report:
---------------------------------

 Operating Report for the Model mpc_cstr_plant.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) mpc_cstr_plant/CSTR/Integrator
      x:           311      dx:      8.12e-11 (0)
(2.) mpc_cstr_plant/CSTR/Integrator1
      x:          8.57      dx:     -6.87e-12 (0)

Inputs:
```

```
----------
(1.) mpc_cstr_plant/CAi
      u:            10
(2.) mpc_cstr_plant/Ti
      u:           298
(3.) mpc_cstr_plant/Tc
      u:           298

Outputs:
----------
(1.) mpc_cstr_plant/T
      y:           311    [-Inf Inf]
(2.) mpc_cstr_plant/CA
      y:          8.57    [-Inf Inf]
```

Obtain linear model at the initial condition.

```
plant = linearize(plant_mdl, op_point);
```

Verify that the linear model is open-loop stable at this condition.

```
eig(plant)
```

```
ans =

   -0.5223
   -0.8952
```

Design MPC Controller for Initial Operating Condition

You design an MPC at the initial operating condition.

```
Ts = 0.5;
```

Specify signal types used in MPC. Assume both reactor temperature and concentration are measurable.

```
plant.InputGroup.UnmeasuredDisturbances = [1 2];
plant.InputGroup.ManipulatedVariables = 3;
plant.OutputGroup.Measured = [1 2];
plant.InputName = {'CAi','Ti','Tc'};
```

```
plant.OutputName = {'T','CA'};
```

Create MPC controller with default prediction and control horizons

```
mpcobj = mpc(plant, Ts);
```

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
   for output(s) y1 and zero weight for output(s) y2
```

Set nominal values in the controller. Note that nominal values for unmeasured disturbance must be zero.

```
mpcobj.Model.Nominal = struct('X', x0, 'U', [0;0;u0(3)], 'Y', y0, 'DX', [0 0]);
```

Set scale factors because plant input and output signals have different orders of magnitude

```
Uscale = [10;30;50];
Yscale = [50;10];
mpcobj.DV(1).ScaleFactor = Uscale(1);
mpcobj.DV(2).ScaleFactor = Uscale(2);
mpcobj.MV.ScaleFactor = Uscale(3);
mpcobj.OV(1).ScaleFactor = Yscale(1);
mpcobj.OV(2).ScaleFactor = Yscale(2);
```

The goal will be to track a specified transition in the reactor concentration. The reactor temperature will be measured and used in state estimation but the controller will not attempt to regulate it directly. It will vary as needed to regulate the concentration. Thus, set its MPC weight to zero.

```
mpcobj.Weights.OV = [0 1];
```

Plant inputs 1 and 2 are unmeasured disturbances. By default, the controller assumes integrated white noise with unit magnitude at these inputs when configuring the state estimator. Try increasing the state estimator signal-to-noise by a factor of 10 to improve disturbance rejection performance.

```
D = ss(getindist(mpcobj));
D.b = eye(2)*10;
```

```
setindist(mpcobj, 'model', D);
```

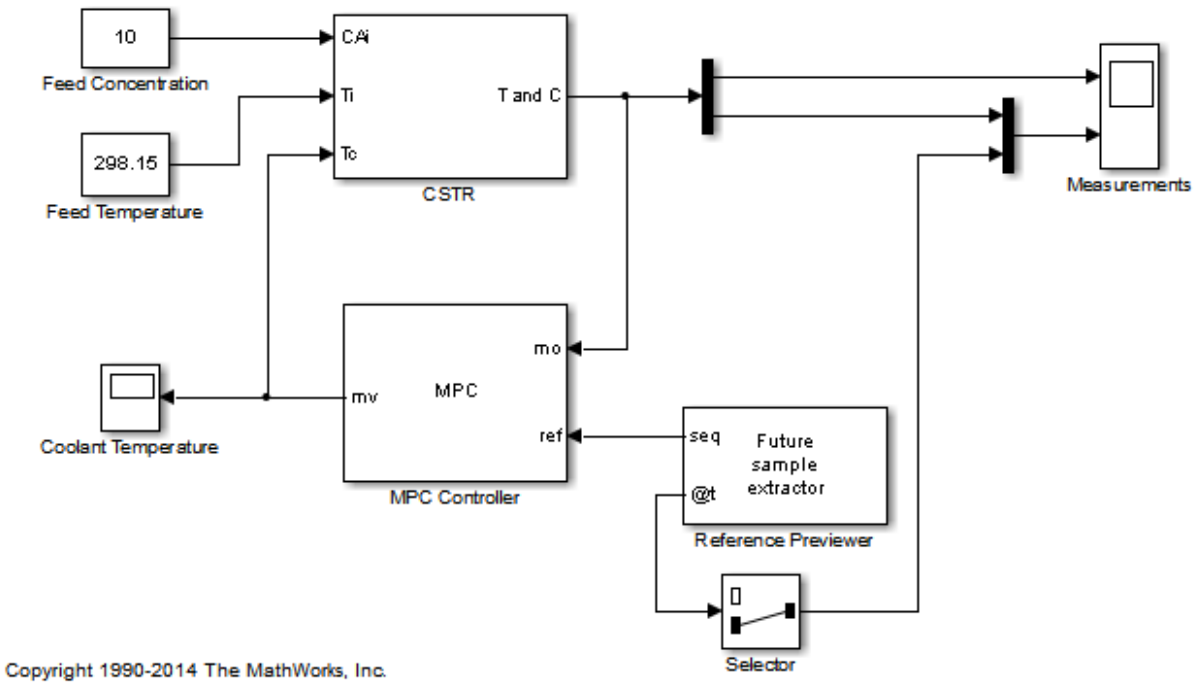
```
-->Converting model to discrete time.
-->The "Model.Disturbance" property of "mpc" object is empty:
   Assuming unmeasured input disturbance #1 is integrated white noise.
   Assuming unmeasured input disturbance #2 is integrated white noise.
   Assuming unmeasured input disturbance #2 is white noise.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

All other MPC parameters are at their default values.

Test the Controller With a Step Disturbance in Feed Concentration

"mpc_cstr_single" contains a Simulink® model with CSTR and MPC Controller blocks in a feedback configuration.

```
mpc_mdl = 'mpc_cstr_single';
open_system(mpc_mdl)
```



Note that the MPC Controller block is configured to look ahead (preview) the setpoint changes in the future, i.e., anticipating the setpoint transition. This generally improves setpoint tracking.

Define a constant setpoint for the output.

```
CSTR_Setpoints.time = [0; 60];
CSTR_Setpoints.signals.values = [y0 y0]';
```

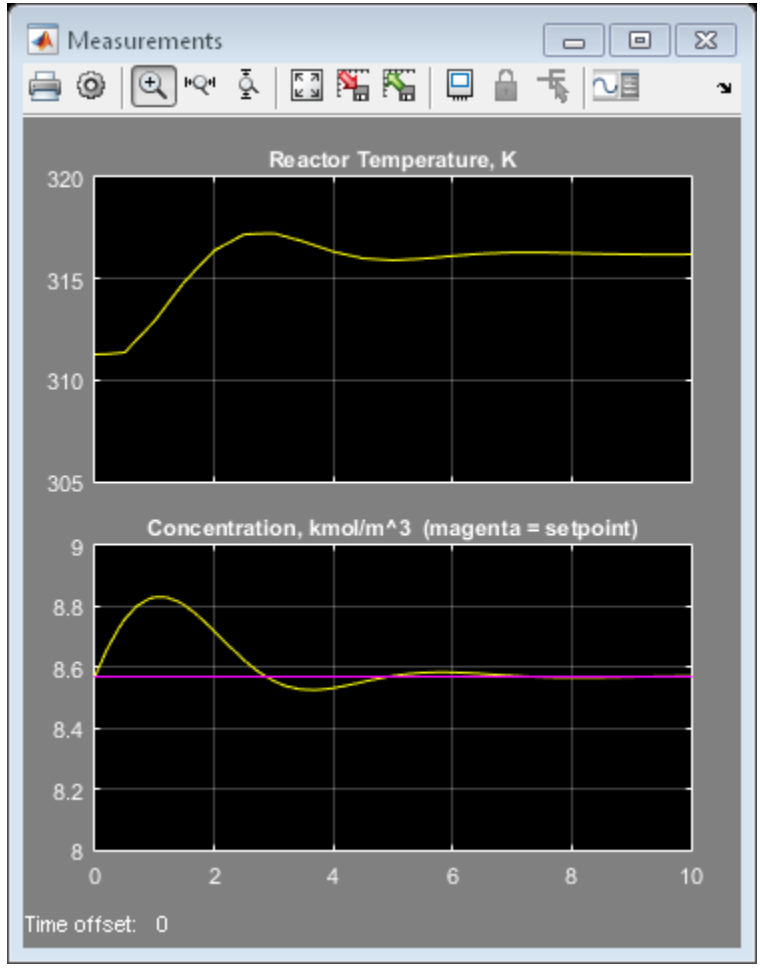
Test the response to a 5% increase in feed concentration.

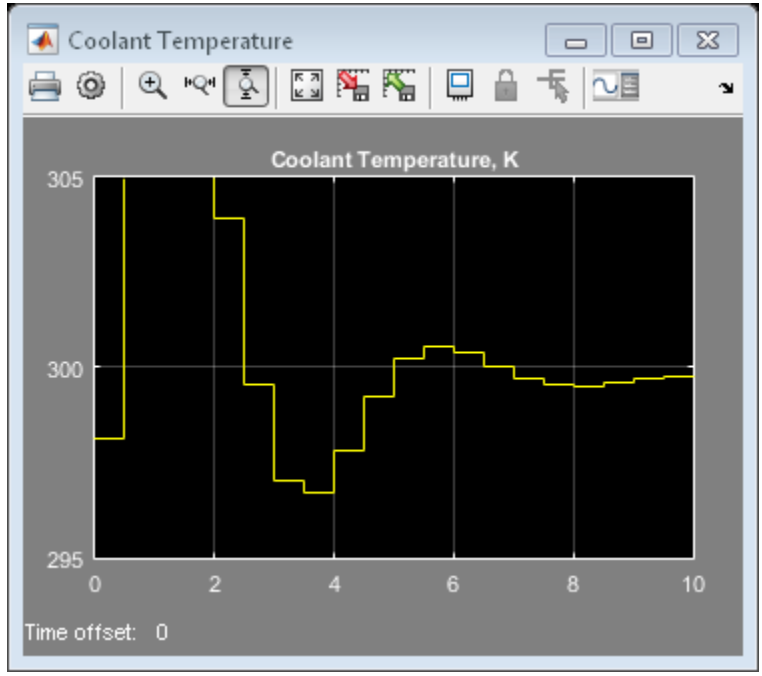
```
set_param([mpc_mdl '/Feed Concentration'], 'Value', '10.5');
```

Set plot scales and simulate the response.

```
open_system([mpc_mdl '/Measurements'])
open_system([mpc_mdl '/Coolant Temperature'])
set_param([mpc_mdl '/Measurements'], 'Ymin', '305~8', 'Ymax', '320~9')
set_param([mpc_mdl '/Coolant Temperature'], 'Ymin', '295', 'Ymax', '305')
sim(mpc_mdl, 10);
```

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


Measurements
Reactor Temperature, K
320
315
310
305
Concentration, kmol/m^3 (magenta = setpoint)
9
8.8
8.6
8.4
8.2
8
0
2
4
6
8
10
Time offset: 0



The closed-loop response is satisfactory.

Simulate Designed MPC Controller Using Full Transition

First, define the desired setpoint transition. After a 10-minute warm-up period, ramp the concentration setpoint downward at a rate of 0.25 per minute until it reaches 2.0 kmol/m^3.

```
CSTR_Setpoints.time = [0 10 11:39]';
CSTR_Setpoints.signals.values = [y0(1)*ones(31,1),[y0(2);y0(2);(y0(2):-0.25:2)';2;2]];
```

Remove the 5% increase in feed concentration used previously.

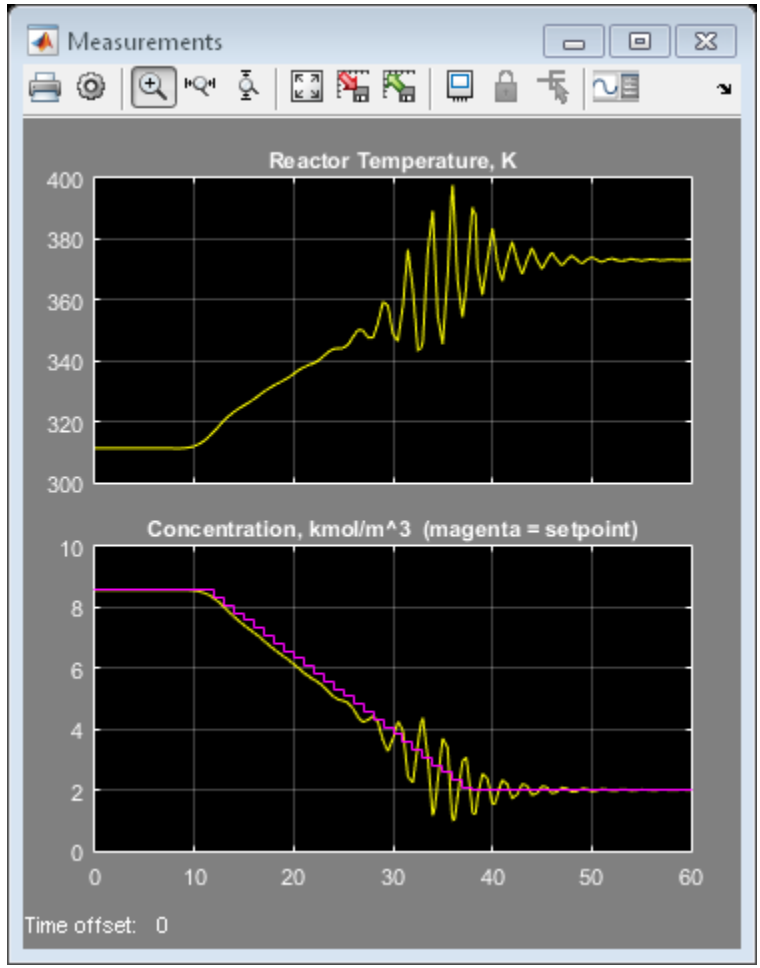
```
set_param([mpc_mdl '/Feed Concentration'], 'Value', '10')
```

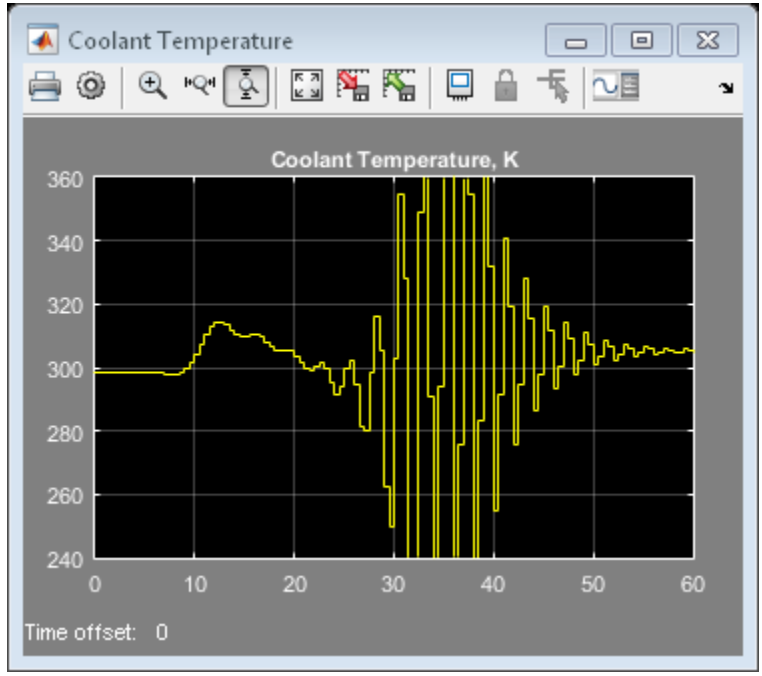
Set plot scales and simulate the response.

```
set_param([mpc_mdl '/Measurements'], 'Ymin', '300~0', 'Ymax', '400~10')
set_param([mpc_mdl '/Coolant Temperature'], 'Ymin', '240', 'Ymax', '360')
```

Simulate model.

```
sim(mpc_mdl, 60)
```





The closed-loop response is unacceptable. Performance along the full transition can be improved if other MPC controllers are designed at different operating conditions along the transition path. In the next two section, two additional MPC controllers are design at intermediate and final transition stages respectively.

Design MPC Controller for Intermediate Operating Condition

Create operating point specification.

```
op = operspec(plant_mdl);
```

Feed concentration is known.

```
op.Inputs(1).u = 10;
op.Inputs(1).Known = true;
```

Feed temperature is known.

```
op.Inputs(2).u = 298.15;
```

```
op.Inputs(2).Known = true;
```

Reactor concentration is known

```
op.Outputs(2).y = 5.5;
op.Outputs(2).Known = true;
```

Find steady state operating condition.

```
[op_point, op_report] = findop(plant_mdl,op);
% Obtain nominal values of x, y and u.
x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];
```

```
 Operating Point Search Report:
---------------------------------

 Operating Report for the Model mpc_cstr_plant.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) mpc_cstr_plant/CSTR/Integrator
      x:           339      dx:      3.42e-08 (0)
(2.) mpc_cstr_plant/CSTR/Integrator1
      x:           5.5      dx:     -2.87e-09 (0)

Inputs:
----------
(1.) mpc_cstr_plant/CAi
      u:            10
(2.) mpc_cstr_plant/Ti
      u:           298
(3.) mpc_cstr_plant/Tc
      u:           298    [-Inf Inf]

Outputs:
----------
(1.) mpc_cstr_plant/T
      y:           339    [-Inf Inf]
(2.) mpc_cstr_plant/CA
      y:           5.5    (5.5)
```

Obtain linear model at the initial condition.

```
plant_intermediate = linearize(plant_mdl, op_point);
```

Verify that the linear model is open-loop unstable at this condition.

```
eig(plant_intermediate)
```

```
ans =

    0.4941
   -0.8357
```

Specify signal types used in MPC. Assume both reactor temperature and concentration are measurable.

```
plant_intermediate.InputGroup.UnmeasuredDisturbances = [1 2];
plant_intermediate.InputGroup.ManipulatedVariables = 3;
plant_intermediate.OutputGroup.Measured = [1 2];
plant_intermediate.InputName = {'CAi','Ti','Tc'};
plant_intermediate.OutputName = {'T','CA'};
```

Create MPC controller with default prediction and control horizons

```
mpcobj_intermediate = mpc(plant_intermediate, Ts);
```

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
   for output(s) y1 and zero weight for output(s) y2
```

Set nominal values, scale factors and weights in the controller

```
mpcobj_intermediate.Model.Nominal = struct('X', x0, 'U', [0;0;u0(3)], 'Y', y0, 'DX', [0
Uscale = [10;30;50];
Yscale = [50;10];
mpcobj_intermediate.DV(1).ScaleFactor = Uscale(1);
mpcobj_intermediate.DV(2).ScaleFactor = Uscale(2);
```

```
mpcobj_intermediate.MV.ScaleFactor = Uscale(3);
mpcobj_intermediate.OV(1).ScaleFactor = Yscale(1);
mpcobj_intermediate.OV(2).ScaleFactor = Yscale(2);
mpcobj_intermediate.Weights.OV = [0 1];
D = ss(getindist(mpcobj_intermediate));
D.b = eye(2)*10;
setindist(mpcobj_intermediate, 'model', D);
```

```
-->Converting model to discrete time.
-->The "Model.Disturbance" property of "mpc" object is empty:
   Assuming unmeasured input disturbance #1 is integrated white noise.
   Assuming unmeasured input disturbance #2 is integrated white noise.
   Assuming unmeasured input disturbance #2 is white noise.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on ea
```

Design MPC Controller for Final Operating Condition

Create operating point specification.

```
op = operspec(plant_mdl);
```

Feed concentration is known.

```
op.Inputs(1).u = 10;
op.Inputs(1).Known = true;
```

Feed temperature is known.

```
op.Inputs(2).u = 298.15;
op.Inputs(2).Known = true;
```

Reactor concentration is known

```
op.Outputs(2).y = 2;
op.Outputs(2).Known = true;
```

Find steady state operating condition.

```
[op_point, op_report] = findop(plant_mdl,op);
% Obtain nominal values of x, y and u.
x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];
```

```
 Operating Point Search Report:
---------------------------------

 Operating Report for the Model mpc_cstr_plant.
 (Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.
States:
----------
(1.) mpc_cstr_plant/CSTR/Integrator
      x:           373      dx:      5.57e-11 (0)
(2.) mpc_cstr_plant/CSTR/Integrator1
      x:             2      dx:      -4.6e-12 (0)

Inputs:
----------
(1.) mpc_cstr_plant/CAi
      u:            10
(2.) mpc_cstr_plant/Ti
      u:           298
(3.) mpc_cstr_plant/Tc
      u:           305    [-Inf Inf]

Outputs:
----------
(1.) mpc_cstr_plant/T
      y:           373    [-Inf Inf]
(2.) mpc_cstr_plant/CA
      y:             2    (2)
```

Obtain linear model at the initial condition.

```
plant_final = linearize(plant_mdl, op_point);
```

Verify that the linear model is again open-loop stable at this condition.

```
eig(plant_final)
```

```
ans =

  -1.1077 + 1.0901i
  -1.1077 - 1.0901i
```

Specify signal types used in MPC. Assume both reactor temperature and concentration are measurable.

```
plant_final.InputGroup.UnmeasuredDisturbances = [1 2];
plant_final.InputGroup.ManipulatedVariables = 3;
plant_final.OutputGroup.Measured = [1 2];
plant_final.InputName = {'CAi','Ti','Tc'};
plant_final.OutputName = {'T','CA'};
```

Create MPC controller with default prediction and control horizons

```
mpcobj_final = mpc(plant_final, Ts);
```

```
-->The "PredictionHorizon" property of "mpc" object is empty. Trying PredictionHorizon
-->The "ControlHorizon" property of the "mpc" object is empty. Assuming 2.
-->The "Weights.ManipulatedVariables" property of "mpc" object is empty. Assuming defau
-->The "Weights.ManipulatedVariablesRate" property of "mpc" object is empty. Assuming d
-->The "Weights.OutputVariables" property of "mpc" object is empty. Assuming default 1
   for output(s) y1 and zero weight for output(s) y2
```

Set nominal values, scale factors and weights in the controller

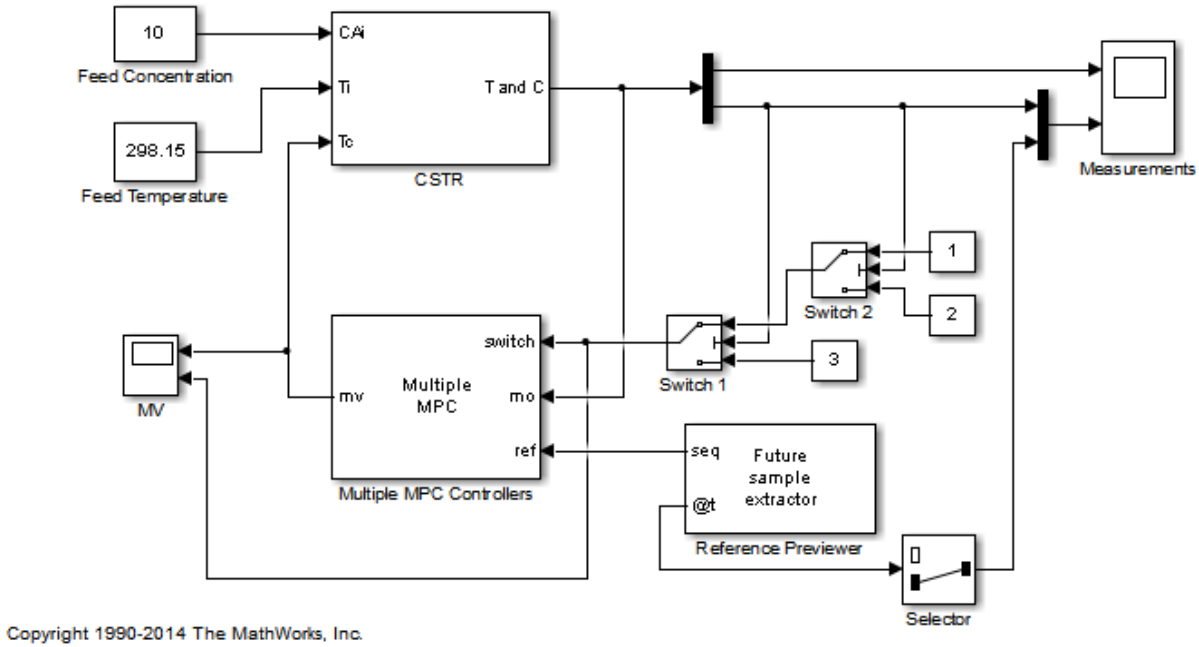
```
mpcobj_final.Model.Nominal = struct('X', x0, 'U', [0;0;u0(3)], 'Y', y0, 'DX', [0 0]);
Uscale = [10;30;50];
Yscale = [50;10];
mpcobj_final.DV(1).ScaleFactor = Uscale(1);
mpcobj_final.DV(2).ScaleFactor = Uscale(2);
mpcobj_final.MV.ScaleFactor = Uscale(3);
mpcobj_final.OV(1).ScaleFactor = Yscale(1);
mpcobj_final.OV(2).ScaleFactor = Yscale(2);
mpcobj_final.Weights.OV = [0 1];
D = ss(getindist(mpcobj_final));
D.b = eye(2)*10;
setindist(mpcobj_final, 'model', D);
```

```
-->Converting model to discrete time.
-->The "Model.Disturbance" property of "mpc" object is empty:
   Assuming unmeasured input disturbance #1 is integrated white noise.
   Assuming unmeasured input disturbance #2 is integrated white noise.
   Assuming unmeasured input disturbance #2 is white noise.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```

Control the CSTR Plant With the Multiple MPC Controllers Block

The following model uses the Multiple MPC Controllers block to implement three MPC controllers across the operating range.

```
mmpc_mdl = 'mpc_cstr_multiple';
open_system(mmpc_mdl);
```



Note that it has been configured to use the three controllers in a sequence: mpcobj, mpcobj_intermediate and mpcobj_final.

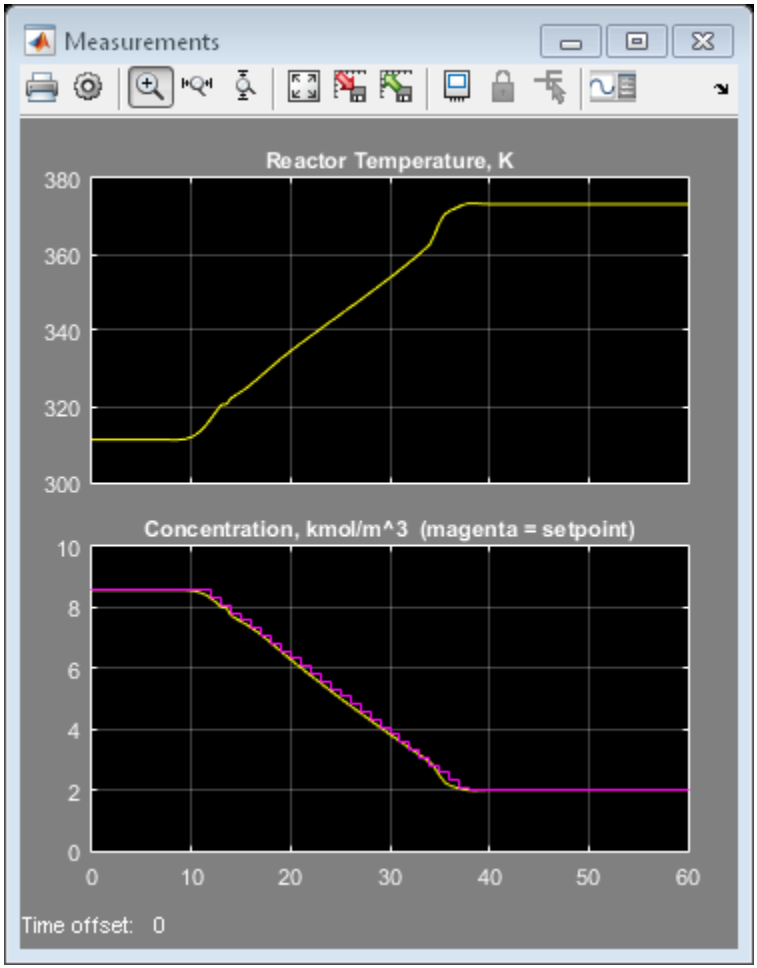
```
open_system([mmpc_mdl '/Multiple MPC Controllers']);
```

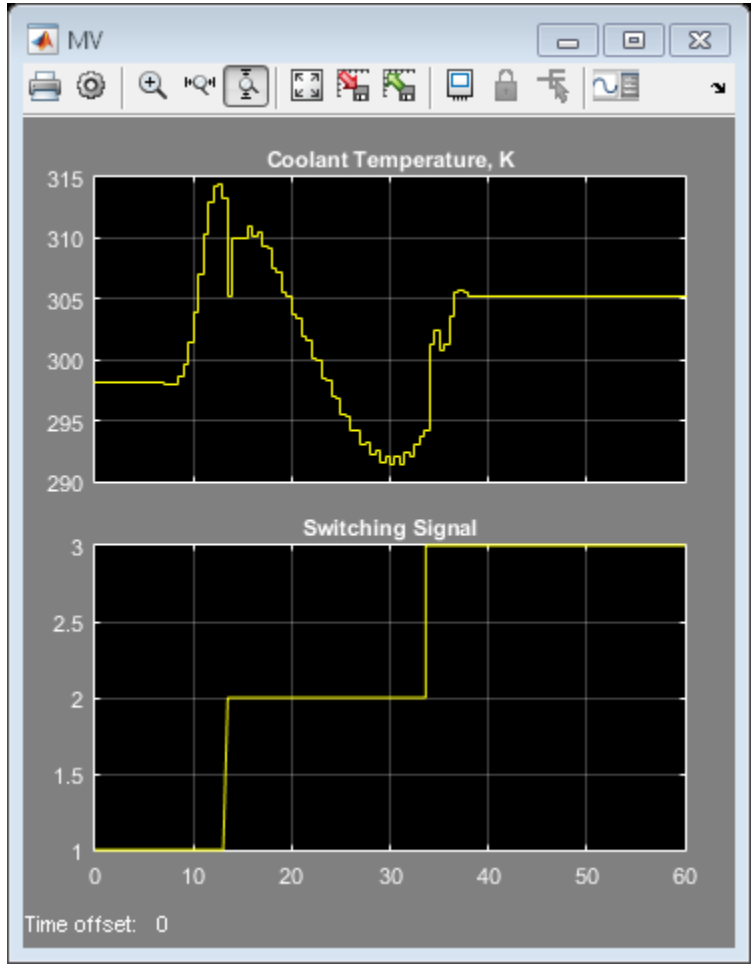
Note also that the two switches specify when to switch from one controller to another. The rules are: 1. If CSTR concentration >= 8, use "mpcobj" 2. If 3 <= CSTR concentration < 8, use "mpcobj_intermediate" 3. If CSTR concentration < 3, use "mpcobj_final"

Simulate with the Multiple MPC Controllers block

```
open_system([mmpc_mdl '/Measurements']);
open_system([mmpc_mdl '/MV']);
```

```
sim(mmpc_mdl)

-->Converting model to discrete time.
   Assuming unmeasured input disturbance #2 is white noise.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting model to discrete time.
   Assuming unmeasured input disturbance #2 is white noise.
   Assuming unmeasured input disturbance #1 is white noise.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```





The transition is now well controlled. The major improvement is in the transition through the open-loop unstable region. The plot of the switching signal shows when controller transitions occur. The MV character changes at these times because of the change in dynamic characteristics introduced by the new prediction model.

```
bdclose(plant_mdl)
bdclose(mpc_mdl)
```

```
bdclose(mmpc_mdl)
```

Related Examples

- “Schedule Controllers at Multiple Operating Points”
- “Coordinate Multiple Controllers at Different Operating Points”
- “Gain Scheduled MPC Control of Mass-Spring System”

More About

- “Design Workflow for Gain Scheduling”

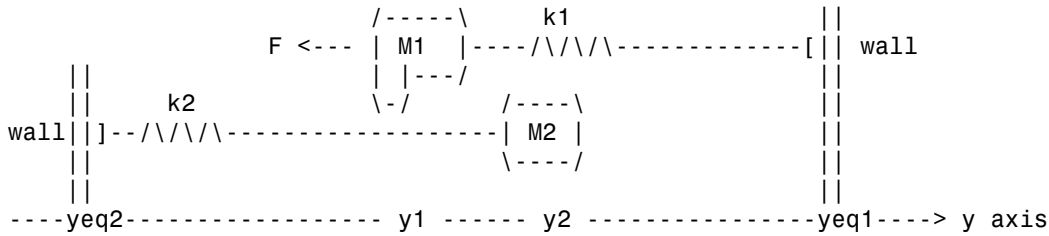
Gain Scheduled MPC Control of Mass-Spring System

This example shows how to use an Multiple MPC Controllers block to implement gain scheduled MPC control of a nonlinear plant.

System Description

The system is composed by two masses M1 and M2 connected to two springs k1 and k2 respectively. The collision is assumed completely inelastic. Mass M1 is pulled by a force F, which is the manipulated variable. The objective is to make mass M1's position y1 track a given reference r.

The dynamics are twofold: when the masses are detached, M1 moves freely. Otherwise, M1+M2 move together. We assume that only M1 position and a contact sensor are available for feedback. The latter is used to trigger switching the MPC controllers. Note that position and velocity of mass M2 are not controllable.



The model is a simplified version of the model proposed in the following reference:

A. Bemporad, S. Di Cairano, I. V. Kolmanovsky, and D. Hrovat, "Hybrid modeling and control of a multibody magnetic actuator for automotive applications," in Proc. 46th IEEE® Conf. on Decision and Control, New Orleans, LA, 2007.

Model Parameters

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

State Space Models

states: position and velocity of mass M1; manipulated variable: pull force F measured disturbance: a constant value of 1 which provides calibrates spring force to the right value measured output: position of mass M1

State-space model of M1 when masses are not in contact.

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

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

State-space model when the two masses are in contact.

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

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

Design MPC Controllers

Common parameters

```
Ts=0.2;     % sampling time
p=20;       % prediction horizon
m=1;        % control horizon
```

Define MPC object for mass M1 detached from M2.

```
MPC1=mpc(sys1,Ts,p,m);
MPC1.Weights.OV=1;

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

Define constraints on the manipulated variable.

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

Define MPC object for mass M1 and M2 stuck together.

```
MPC2=mpc(sys2,Ts,p,m);
MPC2.Weights.OV=1;
```

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

Define constraints on the manipulated variable.

```
MPC2.MV=MPC1.MV;
```

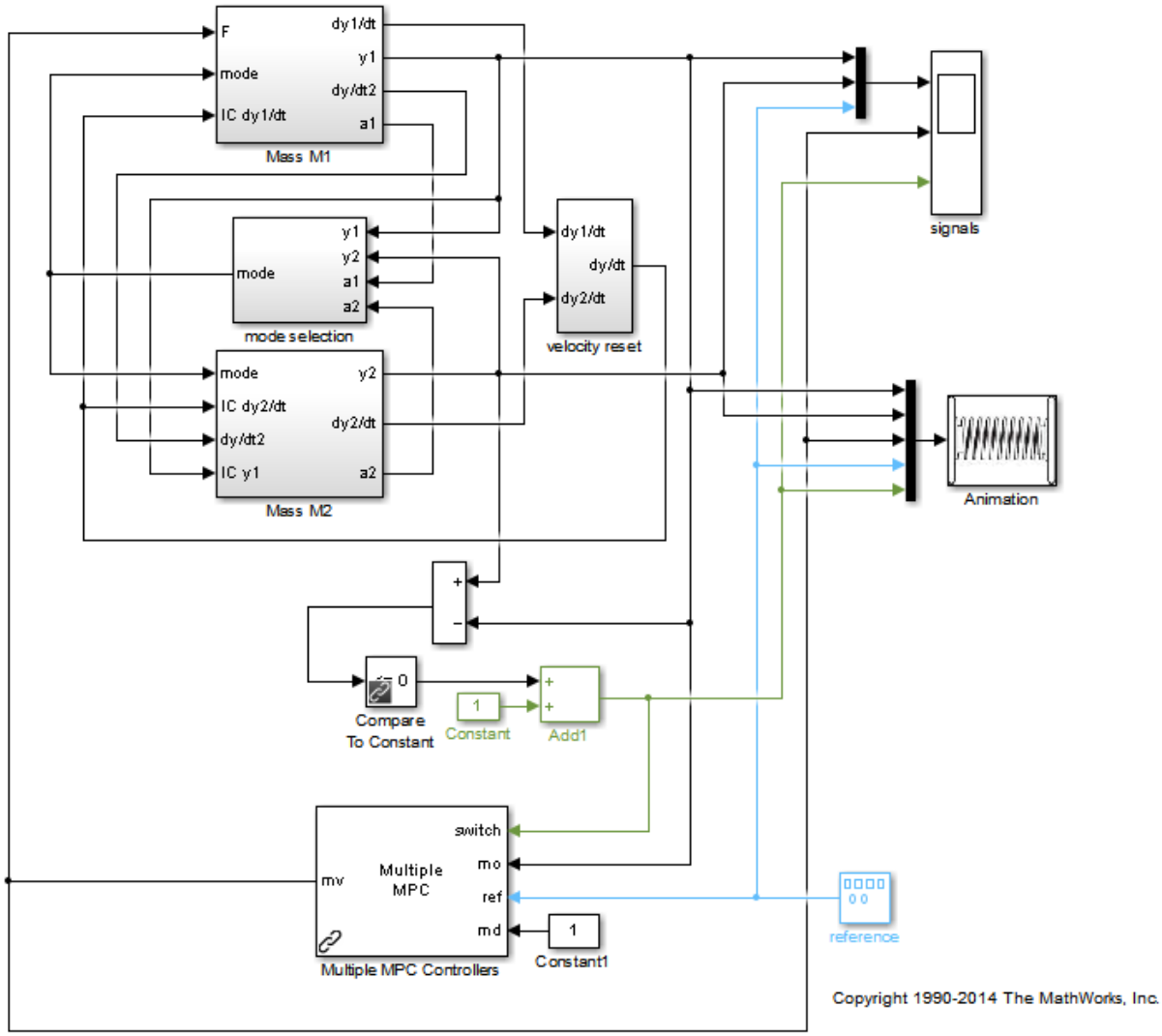
Simulate Gain Scheduled MPC in Simulink®

To run this example, Simulink® is required.

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

Simulate gain scheduled MPC control with Multiple MPC Controllers block.

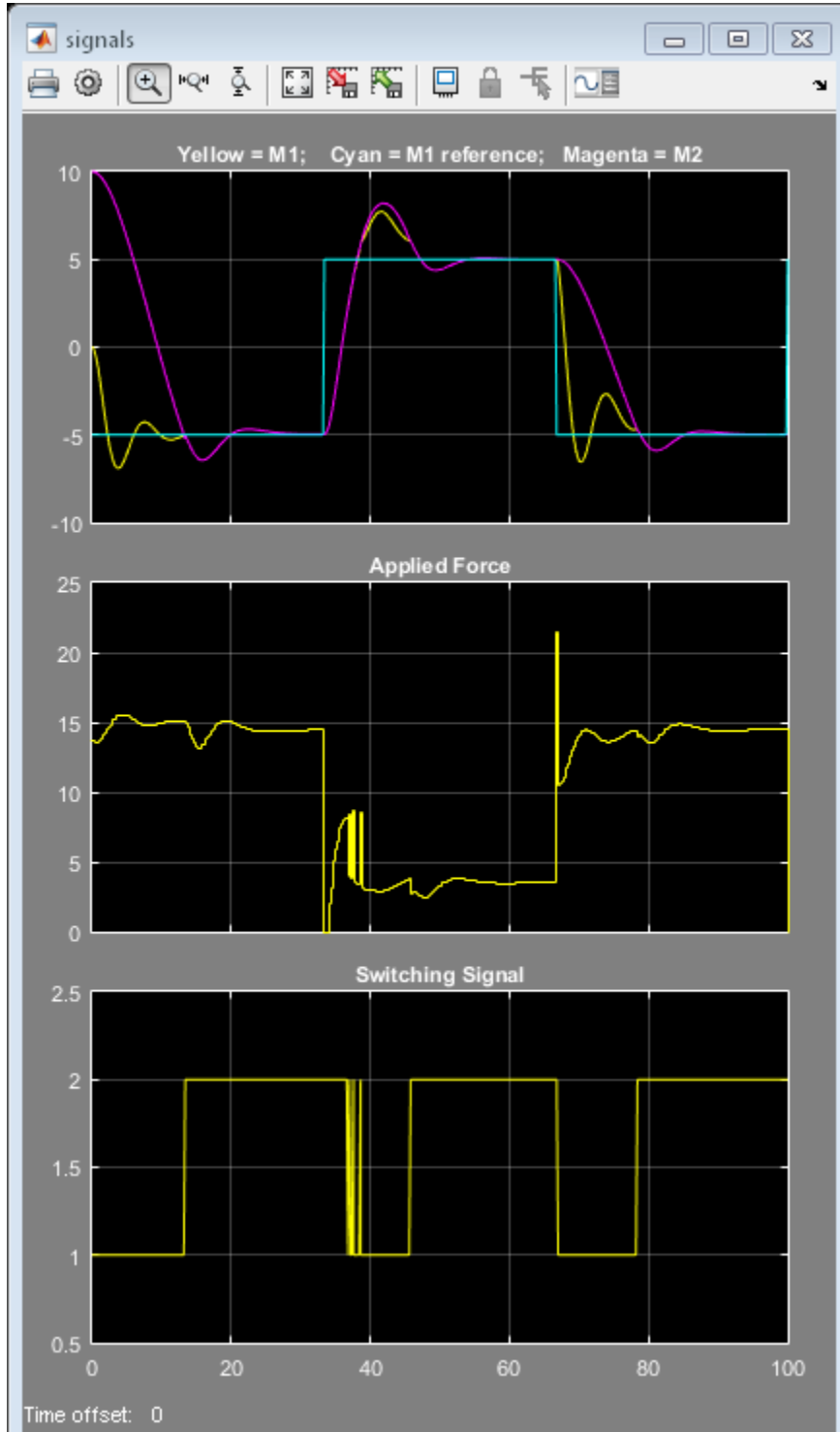
```
y1initial=0;    % Initial positions
y2initial=10;
open_system(mdl);
if exist('animationmpc_switchoff','var') && animationmpc_switchoff
    close_system([mdl '/Animation']);
    clear animationmpc_switchoff
end
```

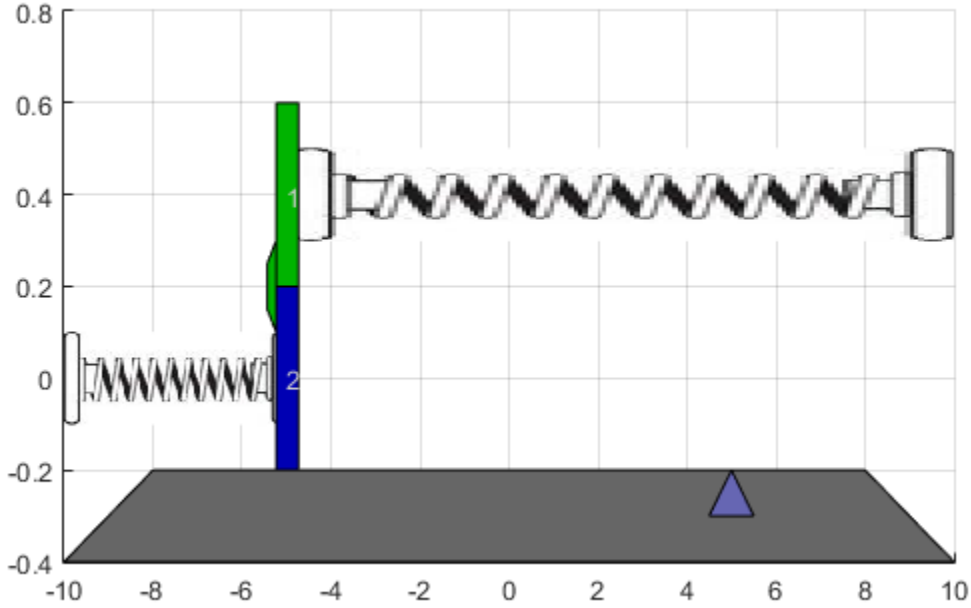



```
disp('Start simulation by switching control between MPC1 and MPC2 ...');
disp('Control performance is satisfactory.');
open_system([mdl '/signals']);
sim(mdl);
```

```
Start simulation by switching control between MPC1 and MPC2 ...
```

```
Control performance is satisfactory.
-->Converting model to discrete time.
-->Integrated white noise added on measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
-->Converting model to discrete time.
-->Integrated white noise added on measured output channel #1.
-->The "Model.Noise" property of the "mpc" object is empty. Assuming white noise on eac
```


signals
Yellow = M1; Cyan = M1 reference; Magenta = M2
10
5
0
-5
-10
Applied Force
25
20
15
10
5
0
Switching Signal
2.5
2
1.5
1
0.5
0
20
40
60
80
100
Time offset: 0

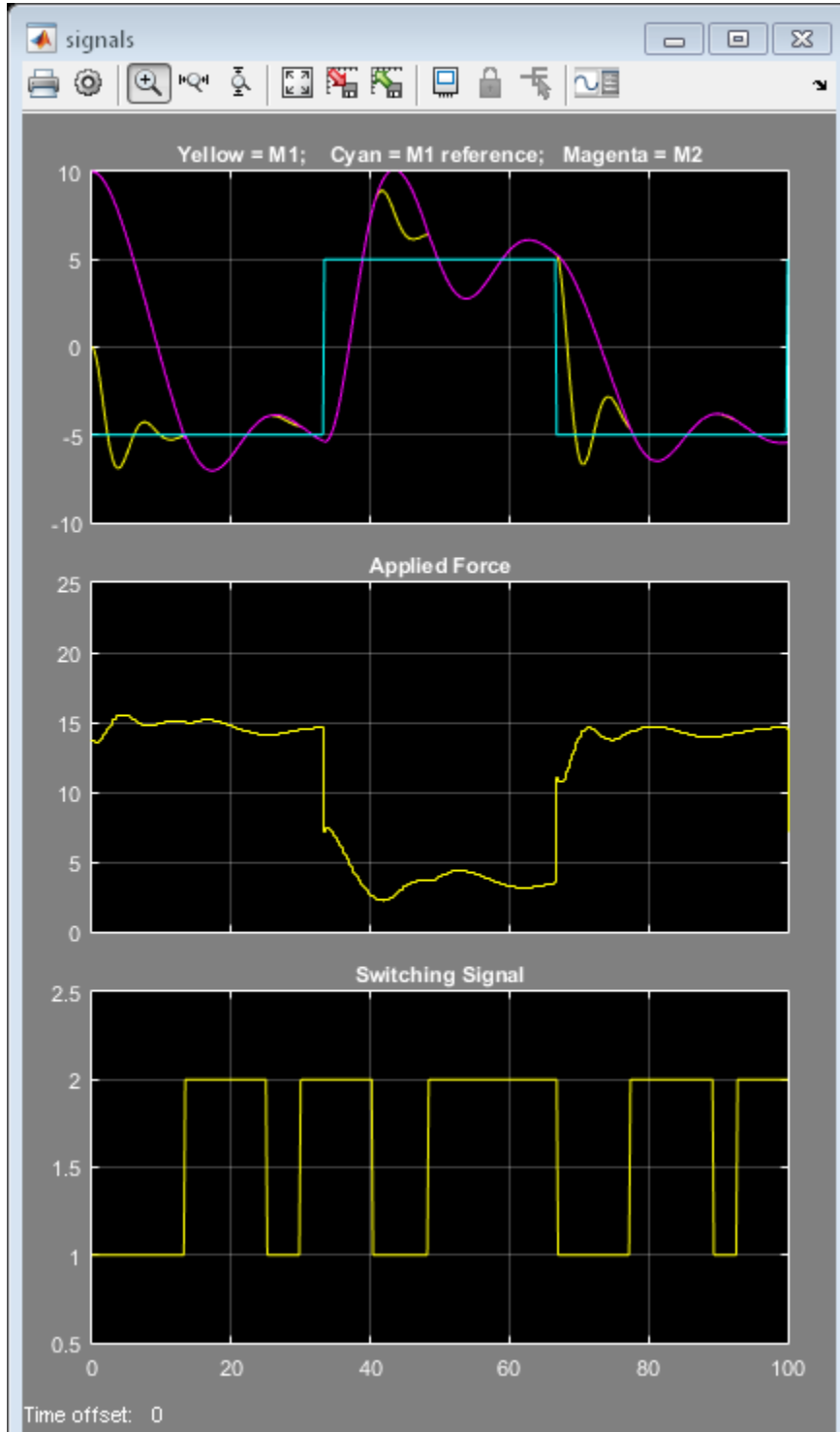


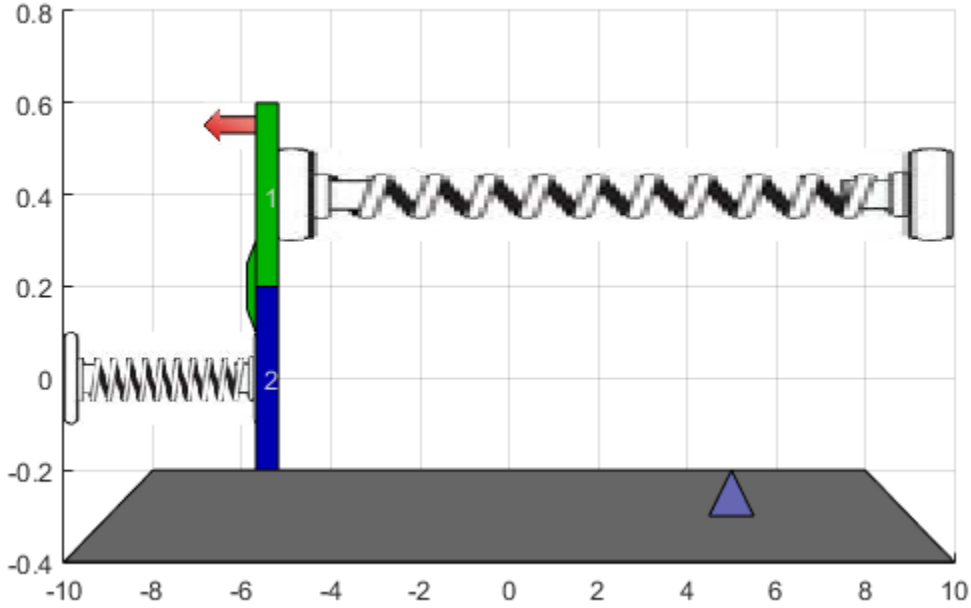
Use of two controllers provides good performance under all conditions.

Repeat Simulation Using MPC1 Only (Assumes Masses Never in Contact)

```
disp('Now repeat simulation by using only MPC1 ...');
disp('When two masses stick together, control performance deteriorates.');
MPC2save=MPC2;
MPC2=MPC1; %#ok<*NASGU>
sim(mdl);
```

```
Now repeat simulation by using only MPC1 ...
When two masses stick together, control performance deteriorates.
```


signals
Yellow = M1; Cyan = M1 reference; Magenta = M2
10
5
0
-5
-10
Applied Force
25
20
15
10
5
0
Switching Signal
2.5
2
1.5
1
0.5
0
20
40
60
80
100
Time offset: 0

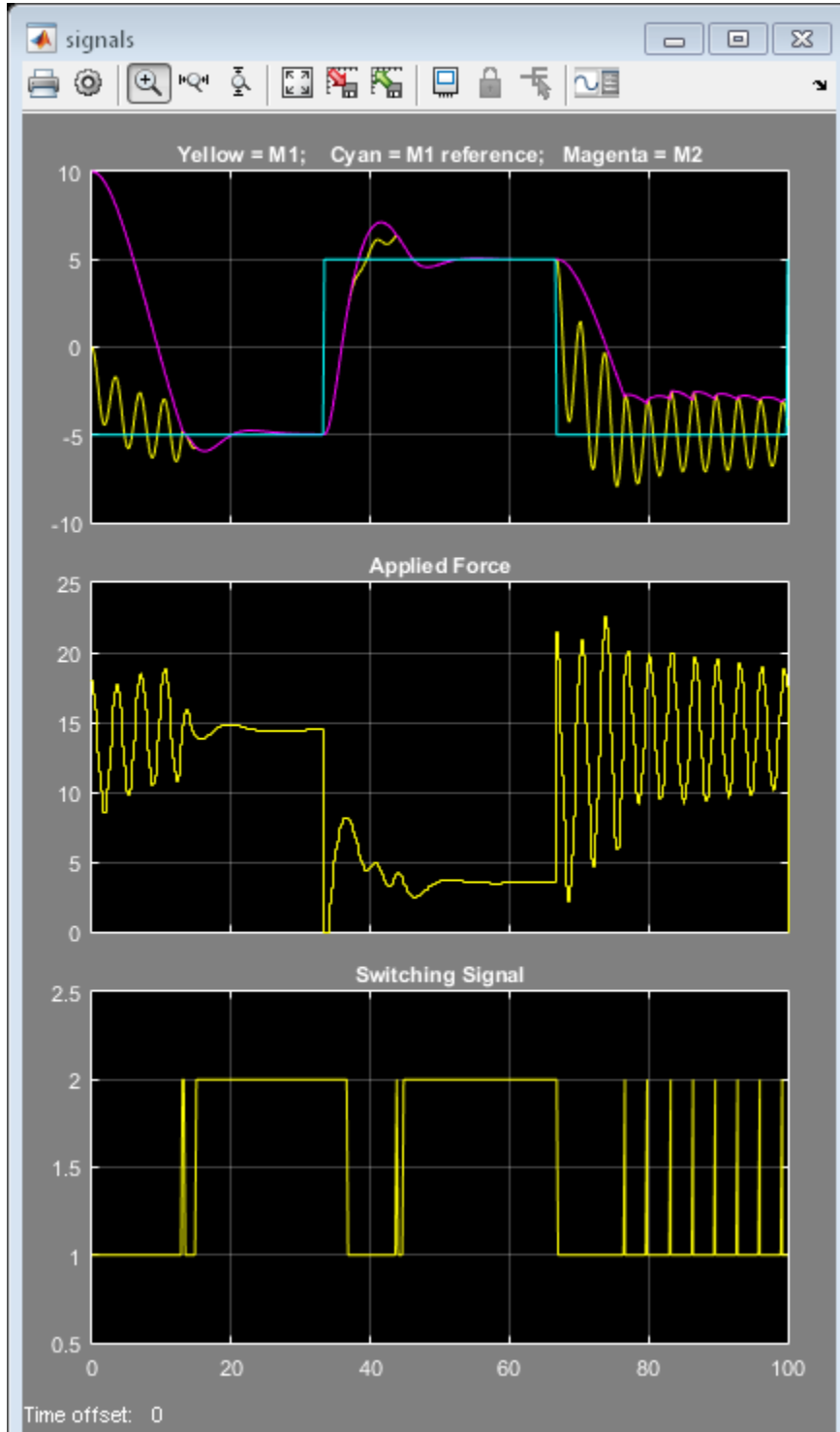


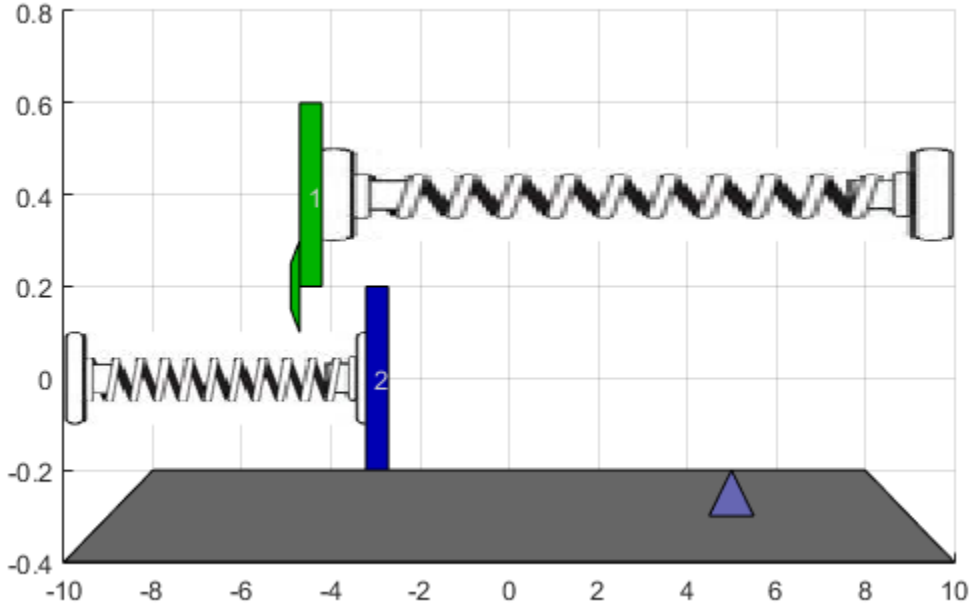
In this case, performance degrades whenever the two masses join.

Repeat Simulation Using MPC2 Only (Assumes Masses Always in Contact)

```
disp('Now repeat simulation by using only MPC2 ...');
disp('When two masses are detached, control performance deteriorates.');
MPC1=MPC2save;
MPC2=MPC1;
sim(mdl);
```

```
Now repeat simulation by using only MPC2 ...
When two masses are detached, control performance deteriorates.
```


signals
Yellow = M1; Cyan = M1 reference; Magenta = M2
10
5
0
-5
-10
Applied Force
25
20
15
10
5
0
Switching Signal
2.5
2
1.5
1
0.5
0
20
40
60
80
100
Time offset: 0



In this case, performance degrades when the masses separate, causing the controller to apply excessive force.

```
bdclose(mdl)
close(findobj('Tag','mpc_switching_demo'))
```

Related Examples

- “Schedule Controllers at Multiple Operating Points”
- “Coordinate Multiple Controllers at Different Operating Points”
- “Gain Scheduled MPC Control of Nonlinear Chemical Reactor”

More About

- “Design Workflow for Gain Scheduling”

8

Reference for the Design Tool GUI

This chapter is the reference manual for the Model Predictive Control Toolbox design tool (graphical user interface).

Working with the Design Tool

In this section...

Opening the MPC Design Tool

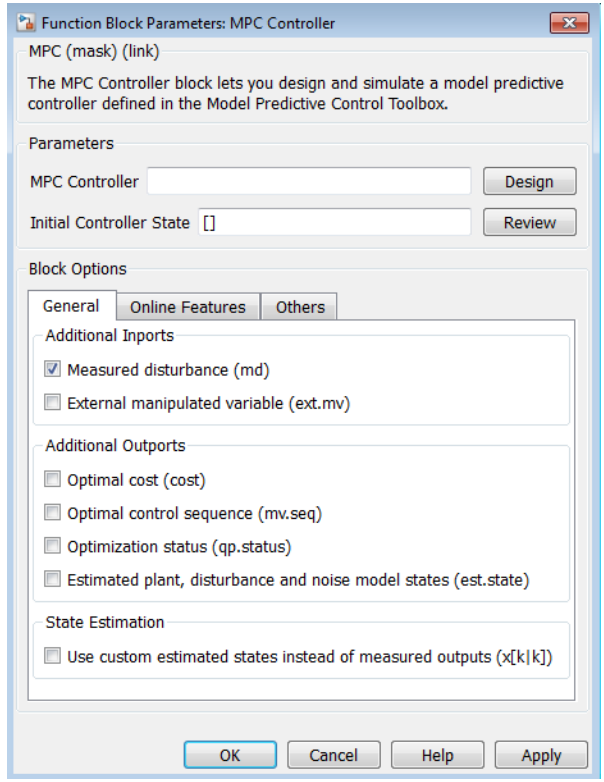
To open the Design Tool in MATLAB, type

```
mpctool
```

The design tool is part of the Control and Estimation Tools Manager. When invoked as shown above, the design tool opens and creates a new *project* named **MPC Design Task**.

If you started the tool previously, the above command makes the tool visible but does not create a new project.

Alternatively, if your Simulink model contains a Model Predictive Controller block, you can double-click the block to obtain its mask (see example below) and click the **Design** button. If the **MPC controller** field is empty, the design tool will create a default controller. Otherwise, it will load the named controller object, which must be in your base workspace. You can then view and modify the controller design.



Creating a New MPC Design Task

To create a new **MPC Design Task**:

1. Select the **Workspace** node in the Control and Estimation Tools Manager.
2. Click **File > New > Task**.
3. In the New Project dialog box, click the **Select** check box for **Model Predictive Control Task: MPC Controller** or **Model Predictive Control Task: Multiple MPC Controllers**.

   Click **OK**.

Menu Bar

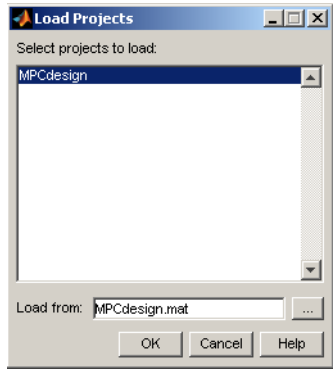
The design tool's menu bar appears whenever you've selected a Model Predictive Control Toolbox project or task in the tree (see “Tree View” on page 8-6). The menu bar's MPC option distinguishes it from other control and estimation tools. See the example below. The following sections describe each menu option.

File Menu

- “Load” on page 8-4
- “Save” on page 8-4
- “Close” on page 8-5

Load

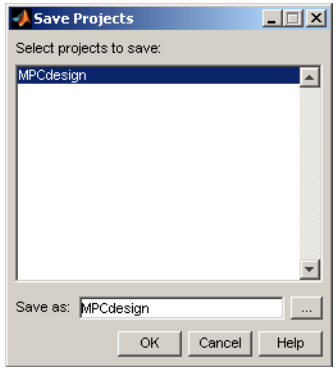
Loads a saved design. A dialog box asks you to specify the MAT-file containing the saved design. If the MAT-file contains multiple projects, you must select the one(s) to be loaded (see example below).



You can also load a design using the toolbar (see “Toolbar” on page 8-6).

Save

Saves a design so you can use it later. The data are saved in a MAT-file. A dialog allows you to specify the file name (see below). If you are working on multiple projects, you can select those to be saved.



You can also select the **Save** option using the toolbar (see “Toolbar” on page 8-6).

Close

Closes the design tool. If you've modified the design, you'll be asked whether or not you want to save it before closing.

MPC Menu

- “Import” on page 8-5
- “Export” on page 8-5
- “Simulate” on page 8-6

Import

You have the following options:

- **Plant model** – Import a plant model using the model import dialog box (see “Importing a Plant Model” on page 8-8).
- **Controller** – Import a controller using the controller import dialog box (see “Importing a Controller” on page 8-12).

Export

Export a controller using the export dialog box (see “Exporting a Controller” on page 8-15). 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 8-6). You can select this option from the keyboard by pressing **Ctrl+R**, or using the toolbar icon (see “Toolbar” on page 8-6).

The **Simulate** option is disabled until your project includes at least one valid simulation scenario.

Toolbar

The toolbar, shown in the following figure, lets you perform the following tasks:

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

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

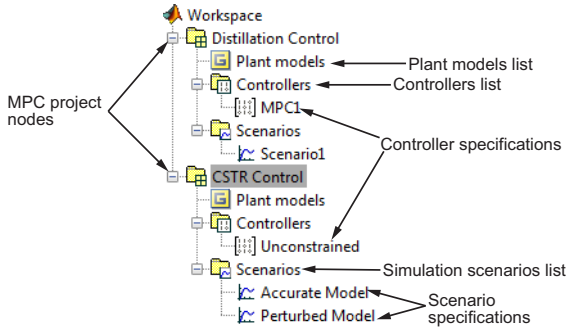
- “Load” on page 8-4
- “Save” on page 8-4
- “Simulate” on page 8-6

The *text output area* is at the bottom of the tool. It displays progress messages and diagnostics. In the above view, the *toggle* button is pushed in, so the text display area appears. If you are working on a small screen, you might use the toggle button to hide the text area, allowing more room to display the controller design.

Tree View

The tree view appears in a frame on the design tool's left-hand side (see example below). When you select one of the tree's *nodes* (by clicking its name or icon) the larger frame to its right shows a dialog pane that allows you to view and edit the specifications associated with that item.

Node Types



The above example shows two Model Predictive Control Toolbox design project nodes, **Distillation Control** and **CSTR Control**, and their subnodes. For more details on each node type, see the following:

- MPC design project/task – See “Signal Definition View” on page 8-16.
- Plant models list – See “Plant Models View” on page 8-20.
- Controllers list – See “Controllers View” on page 8-23.
- Controller specifications – See “Simulation Scenarios List” on page 8-26.
- Scenarios list – See “Simulation Scenario View” on page 8-49.
- Scenario specifications – See “Controller Specifications View” on page 8-29.

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 8-6), 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 8-13 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.

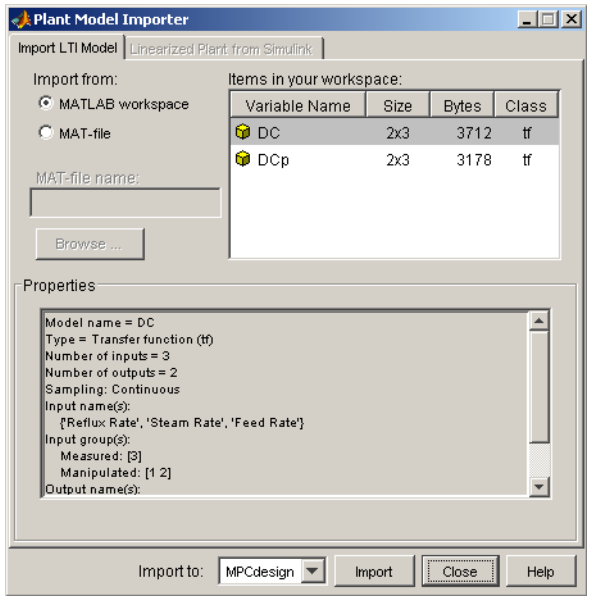
The following sections describe model import options:

- "Import from" on page 8-13
- "Import to" on page 8-14
- "Buttons" on page 8-14
- "Importing a Linearized Plant Model" on page 8-10

Import from

Use these options to set the location from which the model will be imported.

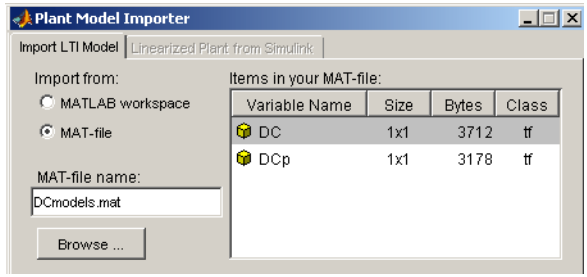
MATLAB workspace



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 8-6, and “Plant Models View” on page 8-20.)

The import dialog box remains visible until you close it so you can import additional models.

Close

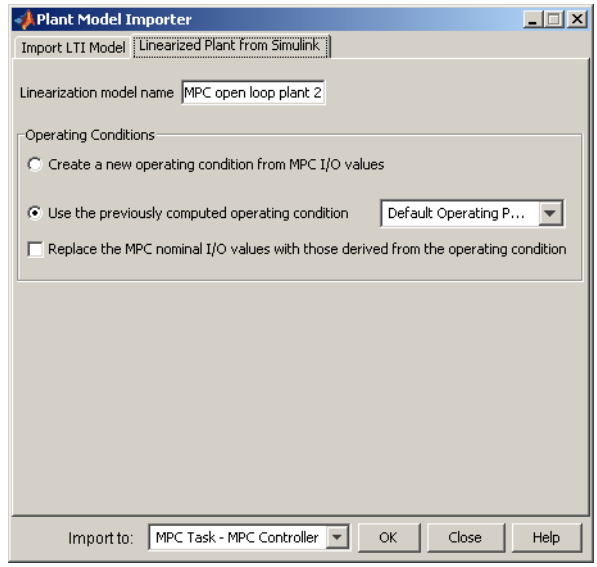
Click **Close** to close the dialog box. You can also click the Close icon on the title bar.

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 8-2.
2. Open the **Plant Model Importer** dialog box (see “Importing a Plant Model” on page 8-8).

3 Click the **Linearized Plant from Simulink** tab (see the following example).

Note If you haven't activated the design tool within Simulink, the **Linearized Plant from Simulink** tab is unavailable.



Linearization Process

If you click **OK**, the design tool uses Simulink Control Design software to create a linearized plant model. It performs the following tasks automatically:

1 Configures the Control and Estimation Tools Manager.

2 Temporarily inserts linearization input and output points in the Simulink model at the inputs and outputs of the MPC block.

3 When the **Create a new operating condition from MPC I/O values** option is selected, the Model Predictive Control Toolbox software temporarily inserts output constraints at the inputs/outputs of the MPC block.

4 Finds a steady state operating condition based on the constraints or uses the specified operating condition.

5 Linearizes the plant model about the operating point.

The linearized plant model appears as a new node under **Plant Models**. For details of the linearization process, refer to the Simulink Control Design documentation.

Linearization Options

You can customize the linearization process in several ways:

- To specify a name for the linearized plant model, enter the name in the **Linearization model name** edit box.
- To use an alternative operating condition, you can:
  - Select one from the menu next to **Use the previously computed operating condition**. This list contains all operating conditions that exist within the current project.
  - Select **Create a new operating condition from MPC I/O values** to compute an operating condition by optimization, using the nominal plant values as constraints. See "Linearize Simulink Models" for an example involving a nonlinear chemical reactor.
- To replace the nominal plant values with the operating point used in the linearization, select the check box next to **Replace the MPC nominal I/O values with those derived from the operating condition**.
- When there are multiple MPC blocks in the Simulink diagram, use the **Import to** menu to select the one that will receive the plant model.

Note The above linearization process automatically identifies the plant's input and output variables according your signal connections in the Simulink model. The controller block does not allow signals corresponding to unmeasured disturbance or unmeasured output variables. Consequently, such variables cannot be included in a model created via the above linearization procedure. If you must include such variables in your controller design, use the Simulink Control Design tool to designate the signals to be used, linearize the plant, and then import this linearized model into the MPC design tool. See "Linearize Simulink Models" for an example of this procedure.

Importing a Controller

To import a controller, do *one* of the following:

- Select the **MPC/Import/Controller** menu option.

- Select the **MPC project/task** node in the tree (see “Tree View” on page 8-6), 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.

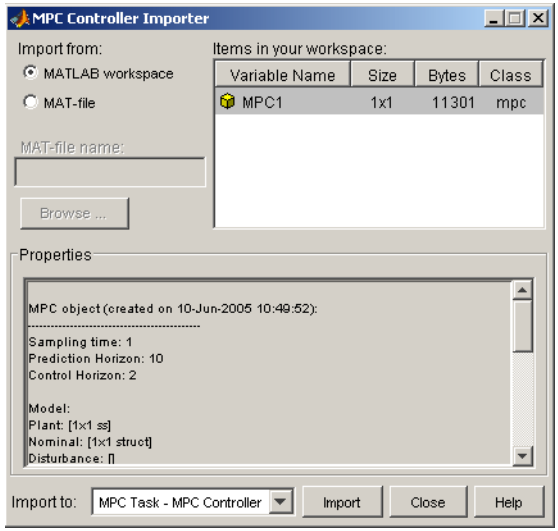
The MPC Controller Importer dialog box opens. The following sections describe its options:

- “Import from” on page 8-13
- “Import to” on page 8-14
- “Buttons” on page 8-14

Import from

Use these options to set the location from which the controller will be imported.

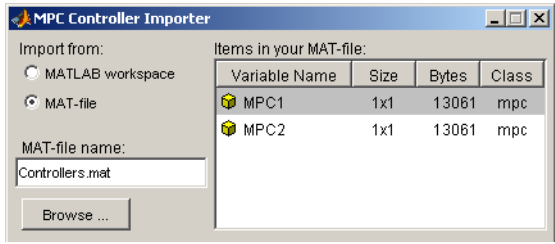
MATLAB Workspace



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 8-6.)

The imported controller contains a plant model, which appears in the **Plant models** list. (See “Plant Models View” on page 8-20.)

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.

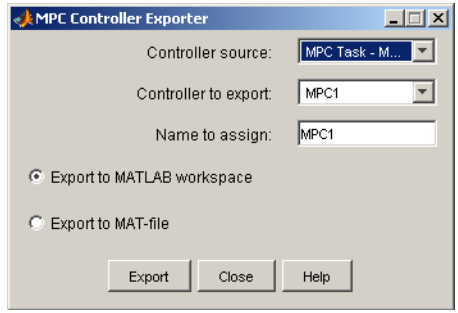
Exporting a Controller

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 8-15
- "Buttons" on page 8-16



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.

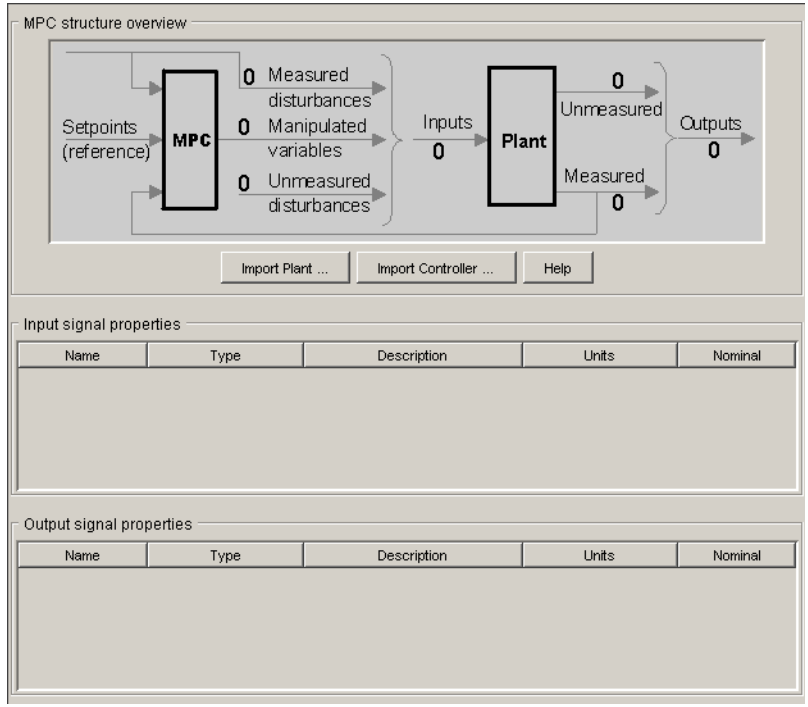
In either case, the dialog box remains visible, allowing you to export additional controllers.

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 8-6). You'll see this view when you open the design tool for the first time. An example appears below.

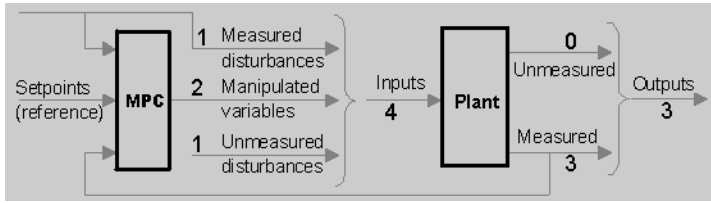


The following sections describe the view's main features:

- “MPC Structure Overview” on page 8-17
- “Buttons” on page 8-18
- “Signal Properties Tables” on page 8-18
- “Right-Click Menu Options” on page 8-20

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 8-8).

Import Controller

Clicking this opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 8-12).

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).

Input signal properties

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

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.).

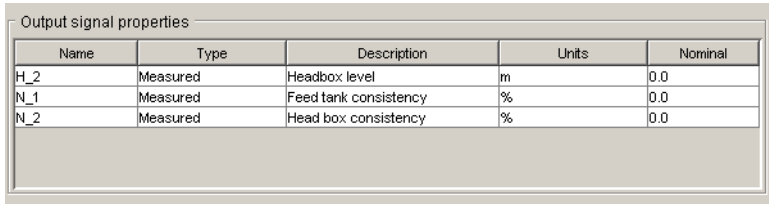
  Measured Disturbance – An independent input whose value is measured and used as a controller input for *feedforward compensation.*

  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.

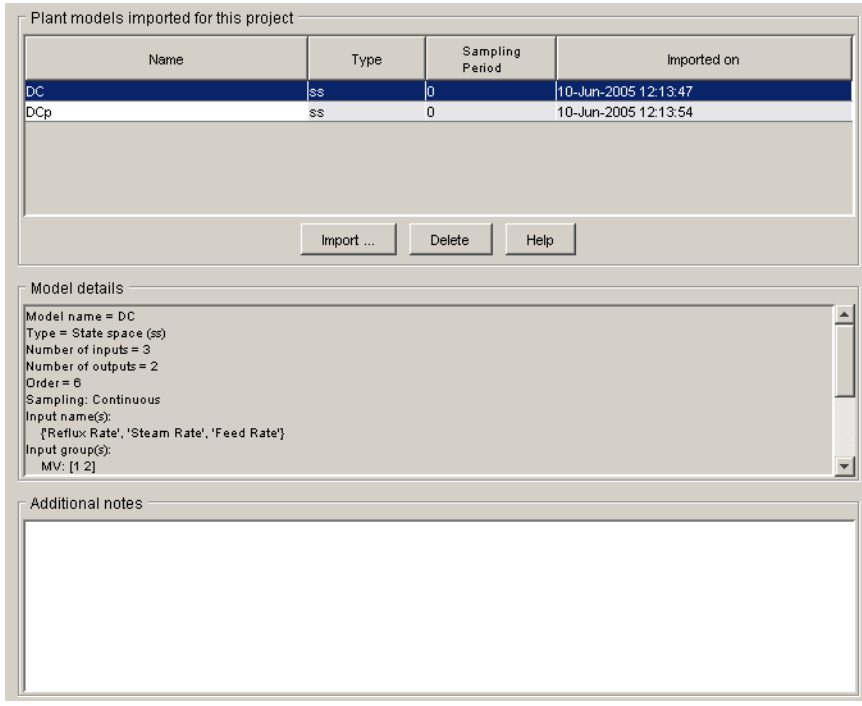
Right-Click Menu Options

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 8-8).
- **Import Controller** – Opens the MPC Controller Importer dialog box (see "Importing a Controller" on page 8-12).
- **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

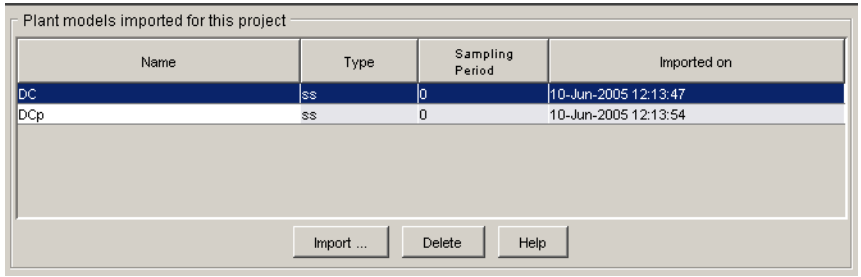
Selecting **Plant models** in the tree displays this view (see example below).



The following sections describe the view's main features:

Plant Models List

This table lists all the plant models you've imported and/or plant models contained in controllers that you've imported. The example below lists two imported models, `DC` and `DCp` .



The **Name** field is editable. Each model must have a unique name. The name you assign here will be used within the design tool only.

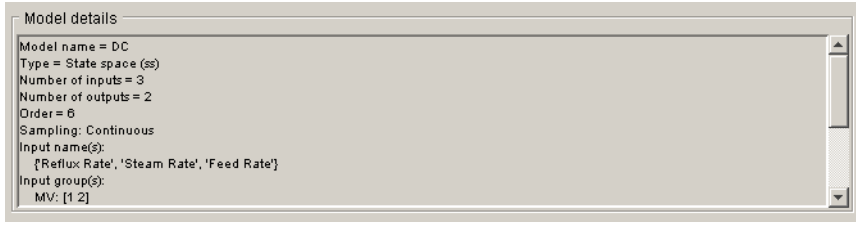
The **Type** field is noneditable and indicates the model's LTI object type (see the Control System Toolbox documentation for a detailed discussion of LTI models).

The **Sampling Period** field is zero for continuous-time models, and a positive real value for discrete-time models.

The **Imported on** field gives the date and time the model was imported.

Model Details

This scrollable viewport shows details of the model currently selected in the plant models list (see “Plant Models List” on page 8-21). 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 8-8).

Delete

Deletes the selected model. If the model is being used elsewhere (i.e., in a controller or scenario), the first model in the list replaces it and a warning message appears.

Right-Click Options

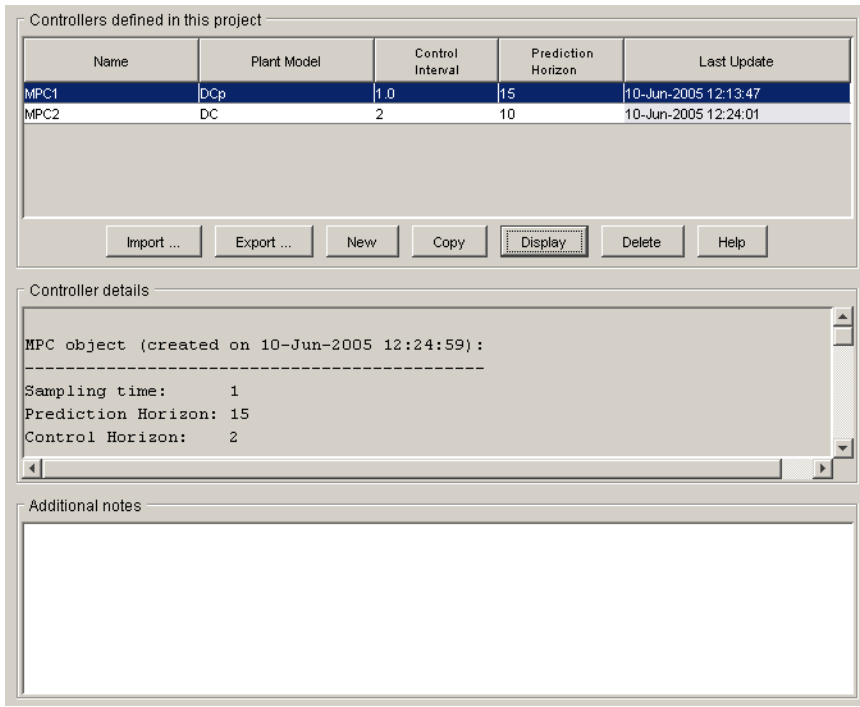
Right-clicking the **Plant models** node causes the following menu option to appear.

Import Model

Opens the Plant Model Importer dialog box (see “Importing a Plant Model” on page 8-8).

Controllers View

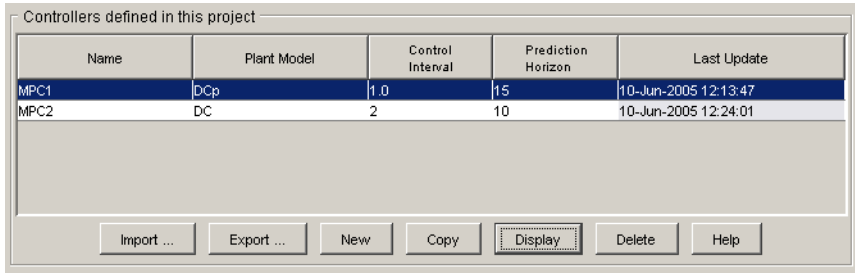
Selecting **Controllers** in the tree displays this view (see example below).



The following sections describe the view's main features:

Controllers List

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



The **Name** field is editable. The name you assign here must be unique. You will refer to it elsewhere in the design tool, e.g., when you use the controller in a simulation scenario. Each listed controller corresponds to a subnode of **Controllers** (see "Tree View" on page 8-6). 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 8-20.)

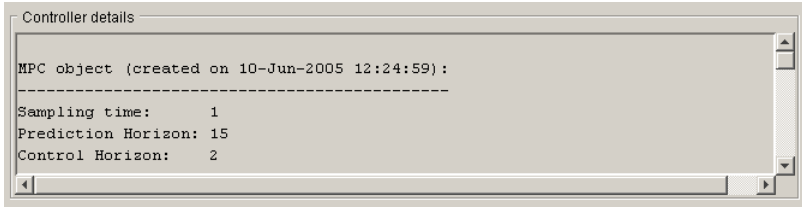
The **Control Interval** field is editable and must be a positive real number. You can also set it in the Controller Specifications view (see "Model and Horizons Tab" on page 8-30 for more details).

The **Prediction Horizon** field is editable and must be a positive, finite integer. You can also set in the Controller Specifications view (see "Model and Horizons Tab" on page 8-30 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 8-24). 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 8-12).

Export

Opens the MPC Controller Exporter dialog box (see “Exporting a Controller” on page 8-15).

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

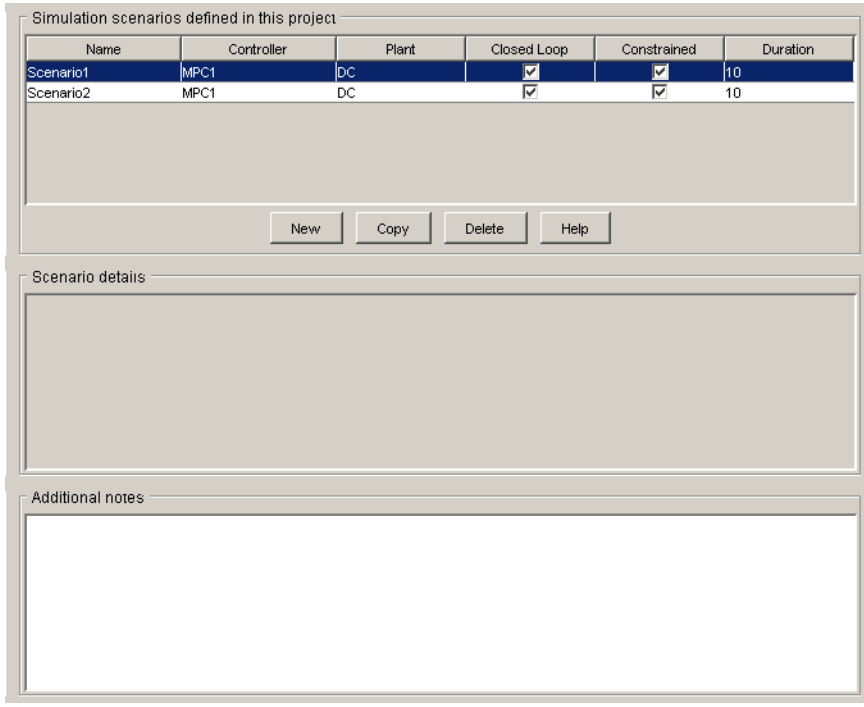
Opens the MPC Controller Importer dialog box (see “Importing a Controller” on page 8-12).

Export Controller

Opens the MPC Controller Exporter dialog box (see “Exporting a Controller” on page 8-15).

Simulation Scenarios List

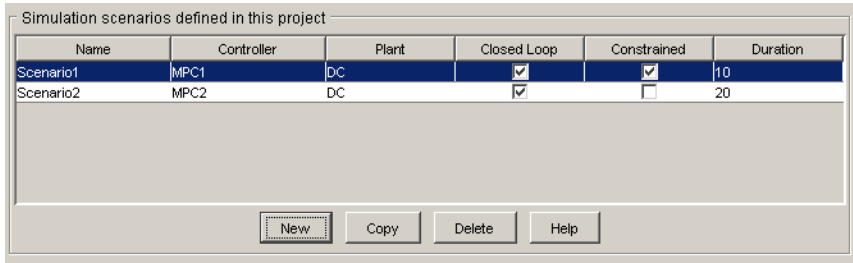
Selecting **Scenarios** in the tree causes this view to appear (see example below).



The following sections describe the view's main features:

Scenarios List

This table lists all the scenarios in your project. The example below lists two, `Scenario1` and `Scenario2` .



The **Name** field is editable. The assigned name must be unique. Each listed scenario corresponds to a subnode of **Scenarios** (see "Tree View" on page 8-6). 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 8-23.) You can also set this using the Scenario Specifications view (for more discussion, see "Simulation Scenario View" on page 8-49).

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 8-20.) You can also set this in the scenario specifications (for more discussion, see "Simulation Scenario View" on page 8-49).

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 8-49).

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 8-49).

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 8-49).

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

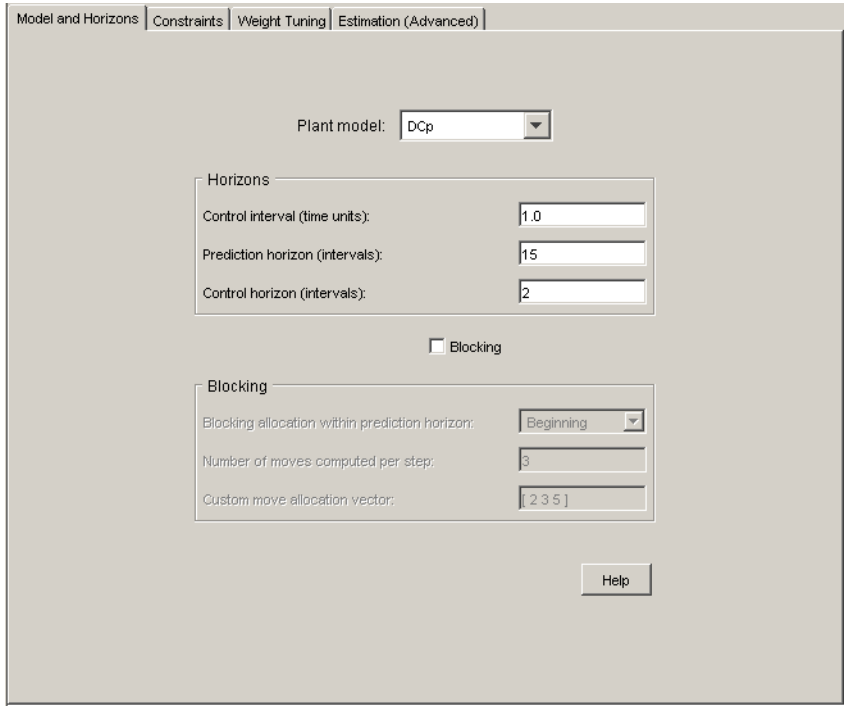
Creates a new scenario specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Controller Specifications View

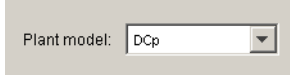
This view appears whenever you select one of your controller nodes (see “Tree View” on page 8-6). It allows you to review and edit controller settings. It consists of four tabs, each devoted to a particular design aspect. All settings have default values.

The following sections describe the view's main features:

Model and Horizons Tab

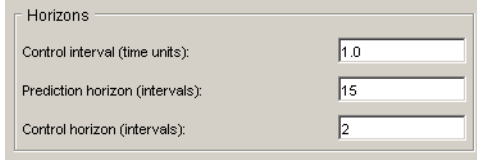


Plant Model



This combo box allows you to specify the plant model the controller uses for its predictions. You can choose any of the plant models you've imported. (See “Importing a Plant Model” on page 8-8.)

Horizons



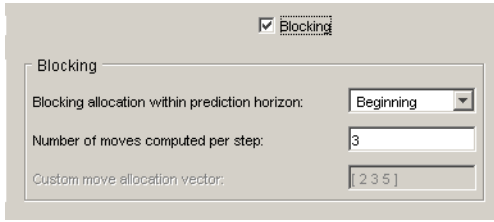
The **Control interval** option sets the elapsed time between successive controller moves. It must be a positive, finite real number. The calculations assume a zero-order hold on the manipulated variables (the signals adjusted by the controller). Thus, these signals are constant between moves.

The **Prediction horizon** option sets the number of *control intervals* over which the controller predicts its outputs when computing controller moves. It must be a positive, finite integer.

The **Control horizon** option sets the number of moves computed. It must be a positive, finite integer, and must not exceed the prediction horizon. If less than the prediction horizon, the final computed move fills the remainder of the prediction horizon.

For more discussion, see "Choosing Sample Time and Horizons" on page 1-5.

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 8-30) with a move pattern determined by the following settings:

- **Blocking allocation within prediction horizon** – Choices are:

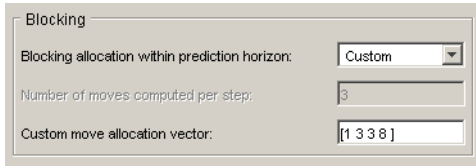
  Beginning – Successive moves at the beginning of the prediction horizon, each with a duration of one control interval.

  Uniform – The prediction horizon is divided by the number of moves and rounded to obtain an integer duration, and each computed move has this duration (the last move extends to fill the prediction horizon).

  End – Successive moves at the end of the prediction horizon, each with a duration of one control interval.

  Custom – You specify the duration of each computed move.

- **Number of moves computed per step** – The number of moves computed when the allocation setting is **Beginning**, **Uniform**, or **End**. Must be a positive integer not exceeding the prediction horizon.
- **Custom move allocation vector** – The duration of each computed move, specified as a row vector. In the example below, there are four moves, the first lasting 1 control interval, the next two lasting 3, and the final lasting 8 for a total of 15. The **Number of moves computed per step** setting is disabled (ignored).



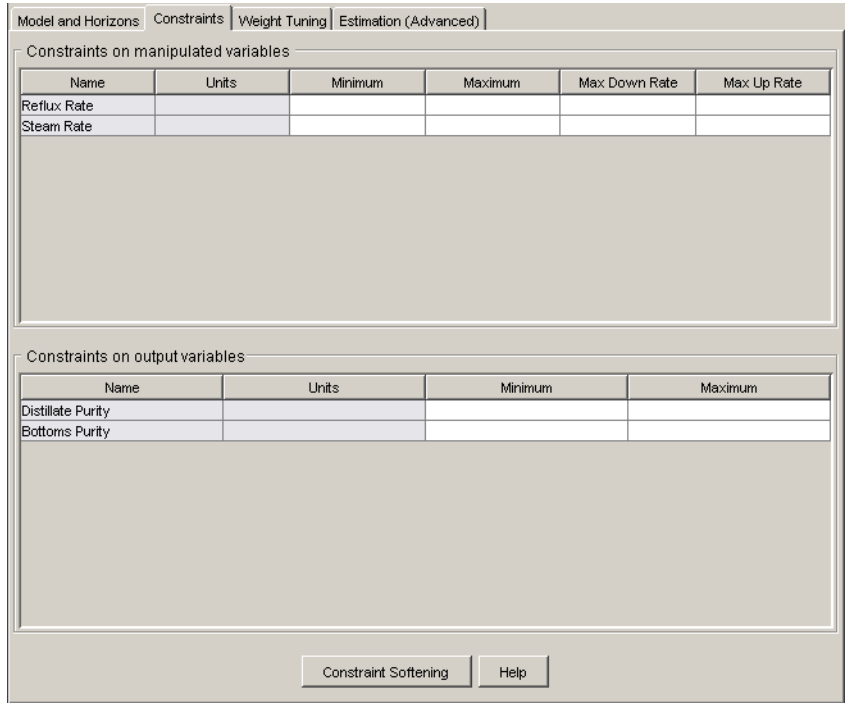
The sum of the vector elements should equal the prediction horizon (15 in this case). If not, the last move is extended or truncated automatically.

Note When **Blocking** is off, the controller uses the **Beginning** allocation with **Number of moves computed per step** equal to the **Control horizon**.

For more discussion, see "Manipulated Variable Blocking" on page 2-26.

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.



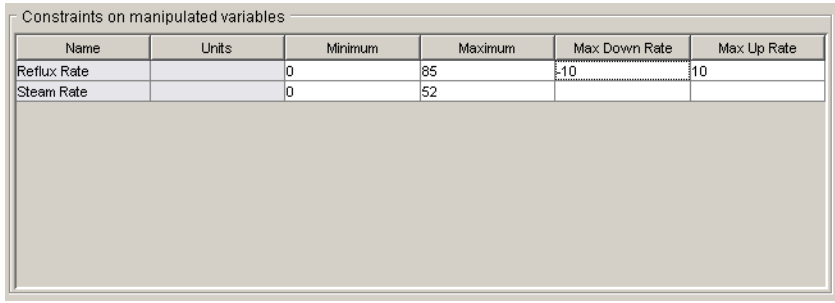
Note If you specify constraints, manipulated variable constraints are hard by default, whereas output variable constraints are soft by default. You can customize this behavior, as discussed in the following sections. For additional information on constraints, see "Constraints" on page 2-7.

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 8-41 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.



The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 8-16. 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.

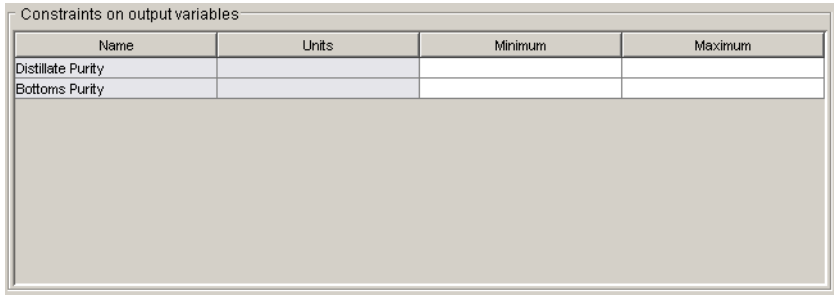
The **Max down rate** and **Max up rate values** set the amount the MV can change *in a single control interval*. The **Max down rate** must be negative or zero. The **Max up rate** must be positive or zero.

Constraint values must be consistent with your nominal values (see "Input Signal Properties" on page 8-18). In other words, each MV's nominal value must satisfy the constraints.

Constraint values must also be self-consistent. For example, an MV's lower bound must not exceed its upper bound.

Constraints on Output Variables

The example below is for an application with two output variables, each represented by a table row.



The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. 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 8-19). In other words, each output's nominal value must satisfy the constraints.

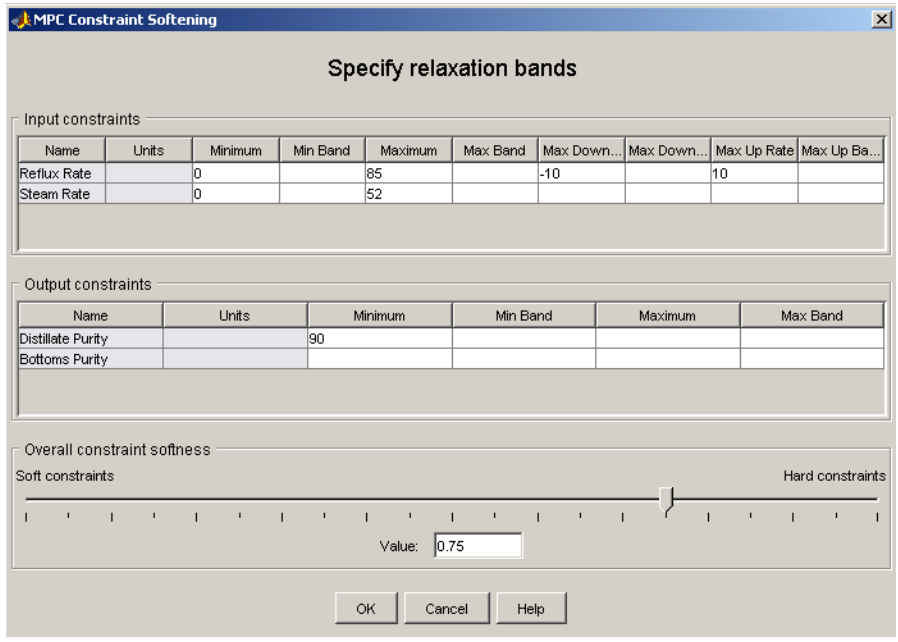
Constraint values must also be self-consistent. For example, an output's lower bound must not exceed its upper bound.

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.

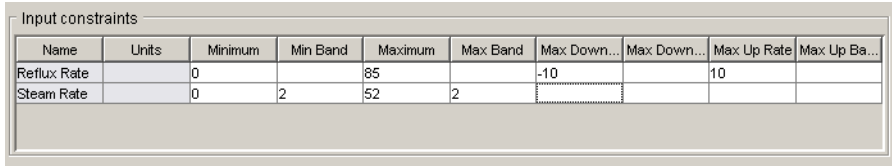


To open this dialog box, click the **Constraint softening** button at the bottom of the **Constraints** tab in the Controller Specification view (see “Constraints Tab” on page 8-32).

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 8-41 for the required format.

Input Constraints

An example input constraint softening specification appears below.



The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. 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 8-34). 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.

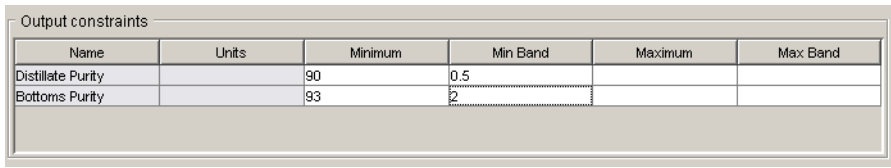
Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

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 8-16. 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 8-34). You can specify them in either location.

The remaining columns specify the *relaxation band* for each constraint. An empty cell is equivalent to 1.0, i.e., a *soft* constraint.

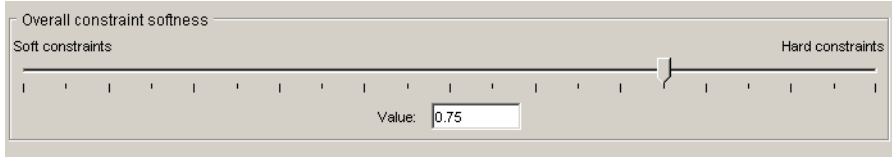
Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

The example above shows a relaxation band of 0.5 mole % for the distillate purity lower bound, and a relaxation band of 2 mole % for the bottoms purity lower bound (the softer of the two constraints).

Note The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band.

Overall Constraint Softness

The relaxation band settings allow you to adjust the hardness/softness of each constraint. You can also soften/harden all constraints simultaneously using the slider at the bottom of the dialog box pane.



You can move the slider or edit the value in the edit box, which must be between 0 and 1.

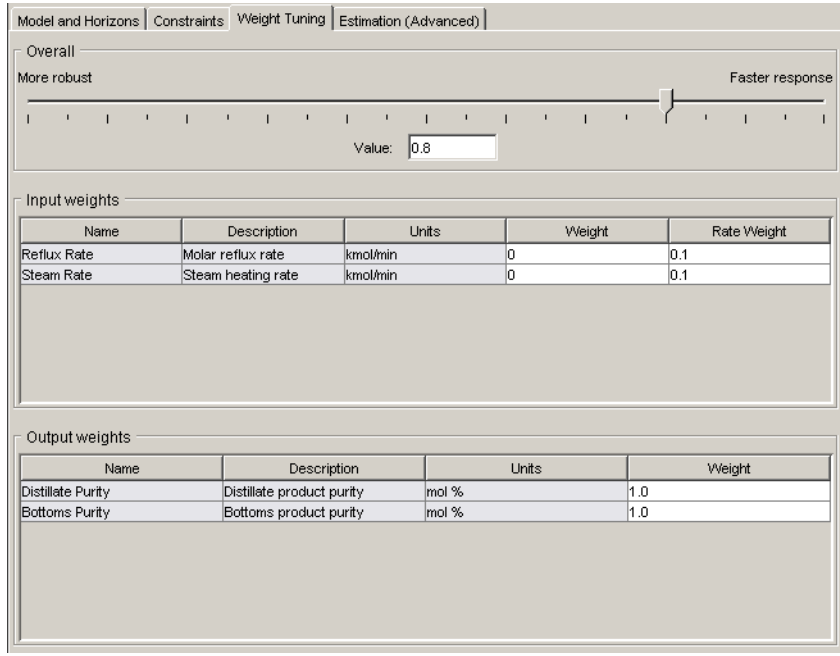
Buttons

OK – Closes the constraint softening dialog box, implementing changes to the tabular entries or the slider setting.

Cancel – Closes the constraint softening dialog box without changing anything.

Weight Tuning Tab

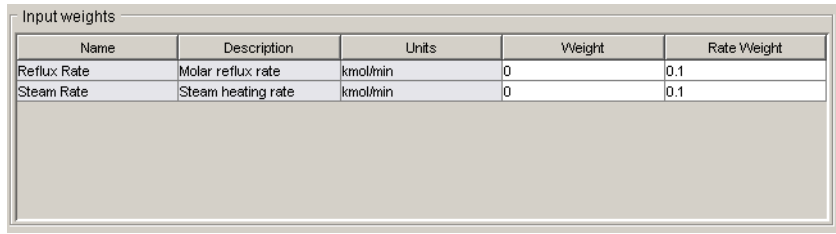
The example below shows the Model Predictive Control Toolbox default tuning weights for an application with two manipulated variables and two outputs.



The following sections discuss the three tab areas in more detail. For additional information, see “Optimization Problem” on page 2-2.

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 8-41 for the required format.

Input Weights



The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. Any changes there apply to the entire design.)

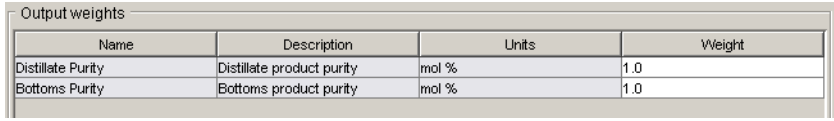
The **Weight** column sets a penalty on deviations of each manipulated variable (MV) from its *nominal value*. The weight must be zero or a positive real number. The default is zero, meaning that the corresponding MV can vary freely provided that it satisfies its constraints (see “Constraints on Manipulated Variables” on page 8-34).

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 8-16. 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



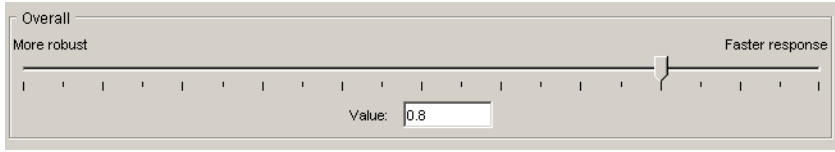
The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. Any changes there apply to the entire design.)

The **Weight** column sets a penalty on deviations of each output variable from its *setpoint* (or *reference*) *value*. The weight must be zero or a positive real number.

A large **Weight** discourages the corresponding output from moving away from its setpoint.

If you don't need to hold a particular output at a setpoint, set its **Weight** to zero. This may be the case, for example, when an output doesn't have a target value and is being used as an indicator variable only.

Overall (Slider Control)



The slider adjusts the weights on all variables simultaneously. Moving the slider to the left increases rate penalties relative to setpoint penalties, which often (but not always!) increases controller robustness. The disadvantage is that disturbance rejection and setpoint tracking become more sluggish.

You can also change the value in the edit box. It must be a real number between 0 and 1. The actual effect is nonlinear. You will generally need to run trials to determine the best setting.

Entering Vectors in Table Cells

In the above examples all constraints and weights were entered as scalars. A scalar entry defines a value that is constant for the entire prediction horizon.

You can also define a constraint or weight that varies with time by entering a *vector*. For the rationale and theoretical basis, see "View and Alter Controller Properties" and "Optimization Problem" on page 2-2.

Enter vectors using the standard MATLAB syntax. For example, `[1, 2, 3]` defines a vector containing three elements, the values 1, 2, and 3.

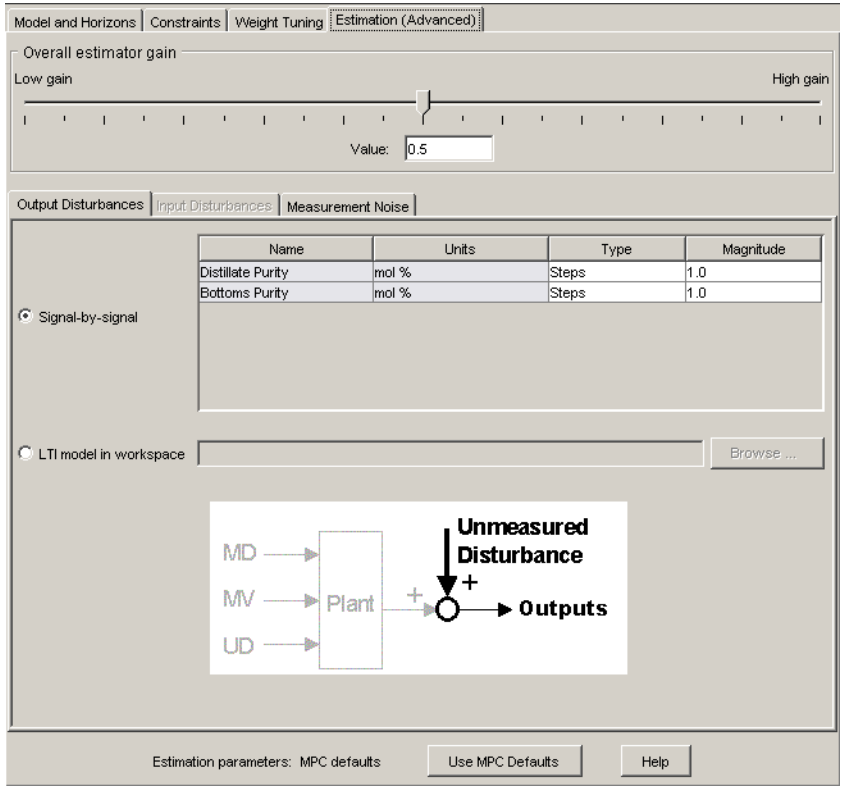
Entries can be either row or column vectors. A MATLAB expression that produces a vector works too. For example, `4*ones(3,1)` would be a valid entry.

If a vector contains fewer than P elements, where P is the horizon length, the controller automatically extends the vector using its last element. For example, if you entered `[1, 2, 3]` and $P = 5$, the vector used in controller calculations would be `[1, 2, 3, 3, 3]`.

Estimation Tab

Use these specifications to shape the controller's response to unmeasured disturbances and measurement noise.

The example below shows Model Predictive Control Toolbox default settings for an application with two output variables and no unmeasured disturbance inputs.



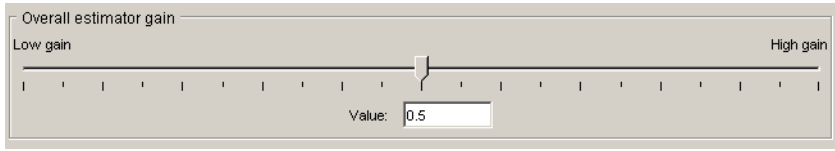
The following sections cover each estimation feature in detail. For additional information, see“Controller State Estimation” on page 2-29.

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



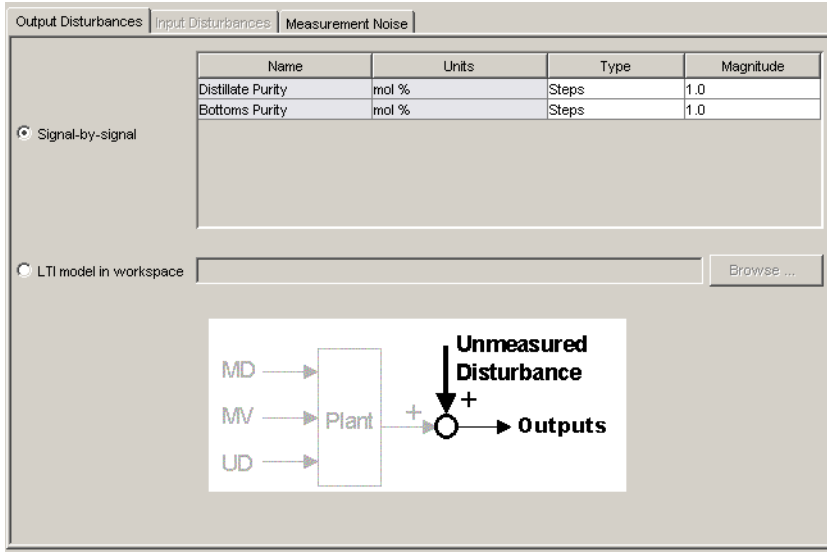
This slider determines the controller's overall disturbance response. As you move the slider to the left, the controller responds less aggressively to unexpected changes in the outputs, i.e., it assumes that such changes are more likely to be caused by measurement noise rather than a *real* disturbance.

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 8-16. Any changes there apply to the entire design.)

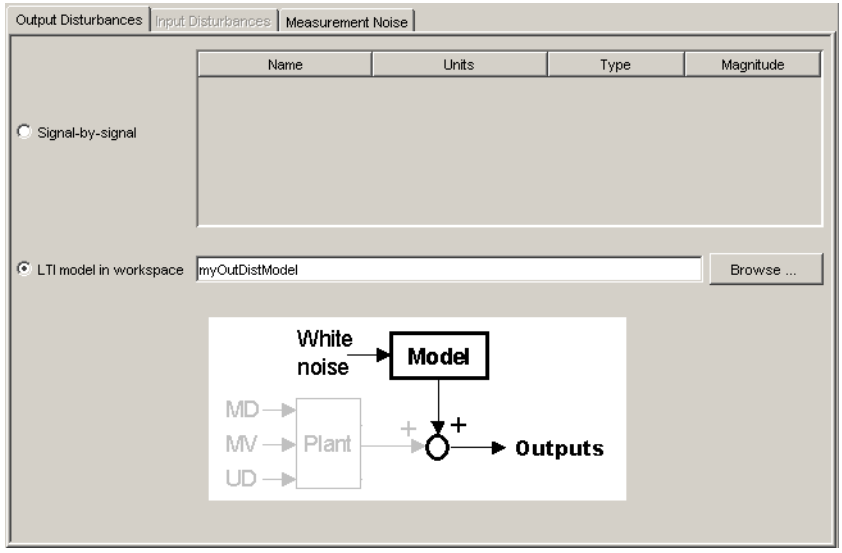
The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **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.

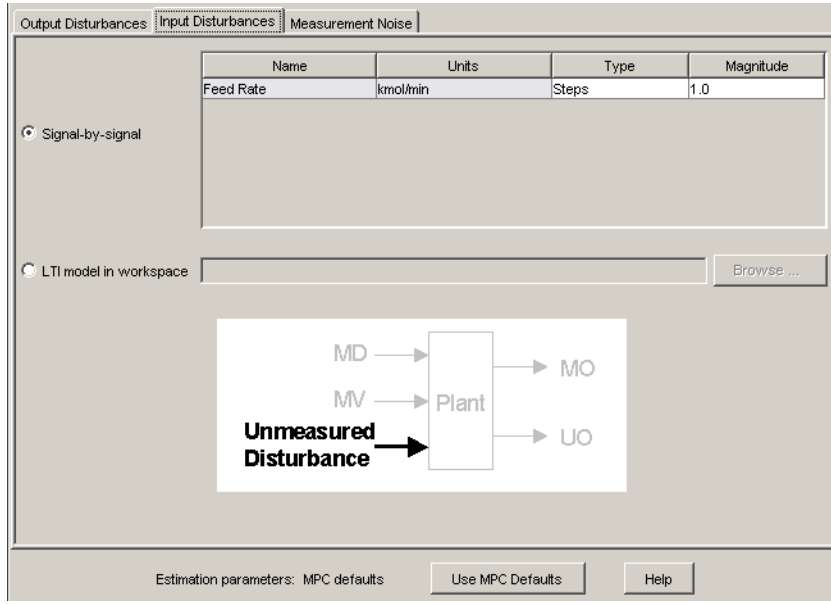
The white noise entering the model is assumed to have unity standard deviation.

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 8-16. Any changes there apply to the entire design.)

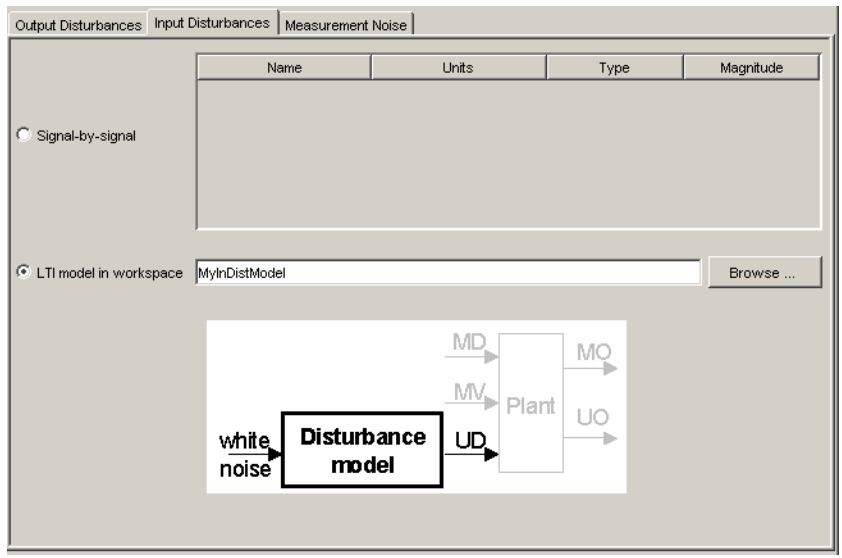
The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **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.



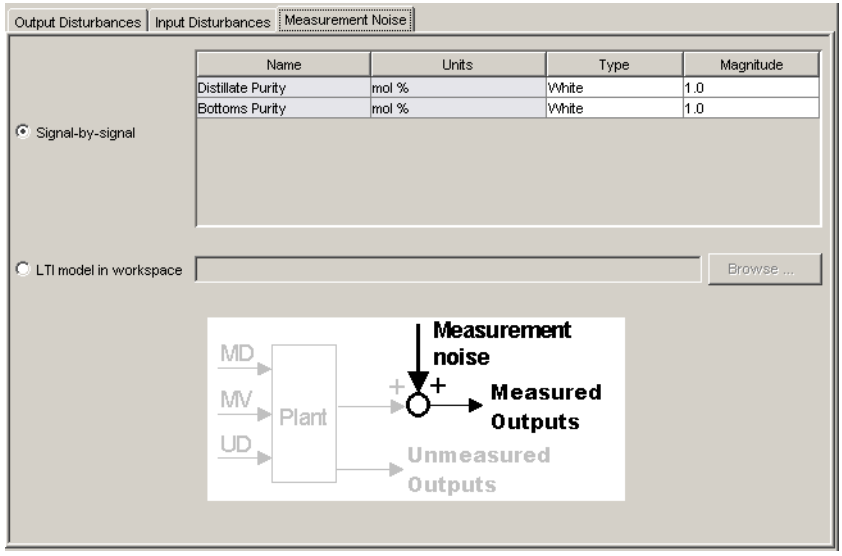
You must specify an LTI disturbance model residing in your workspace. The **Browse** button opens a dialog box listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

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 8-16. Any changes there apply to the entire design.)

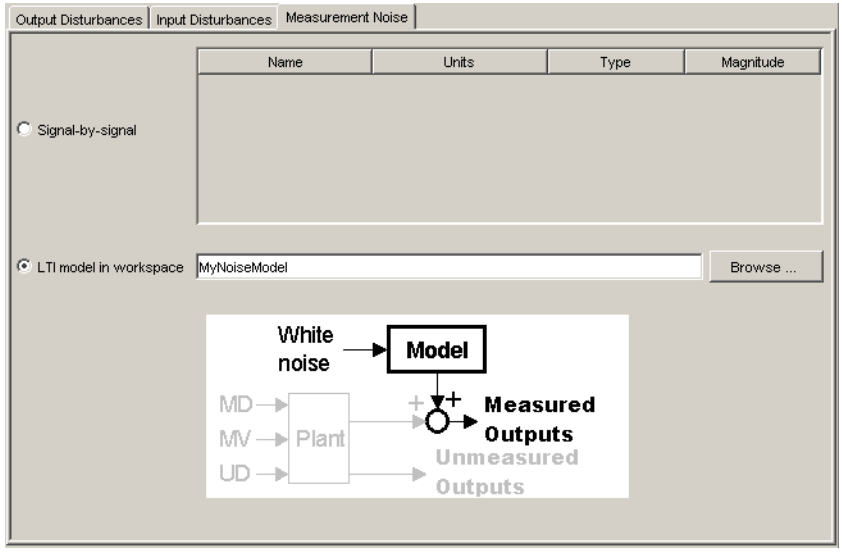
The **Type** column sets the noise character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **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.

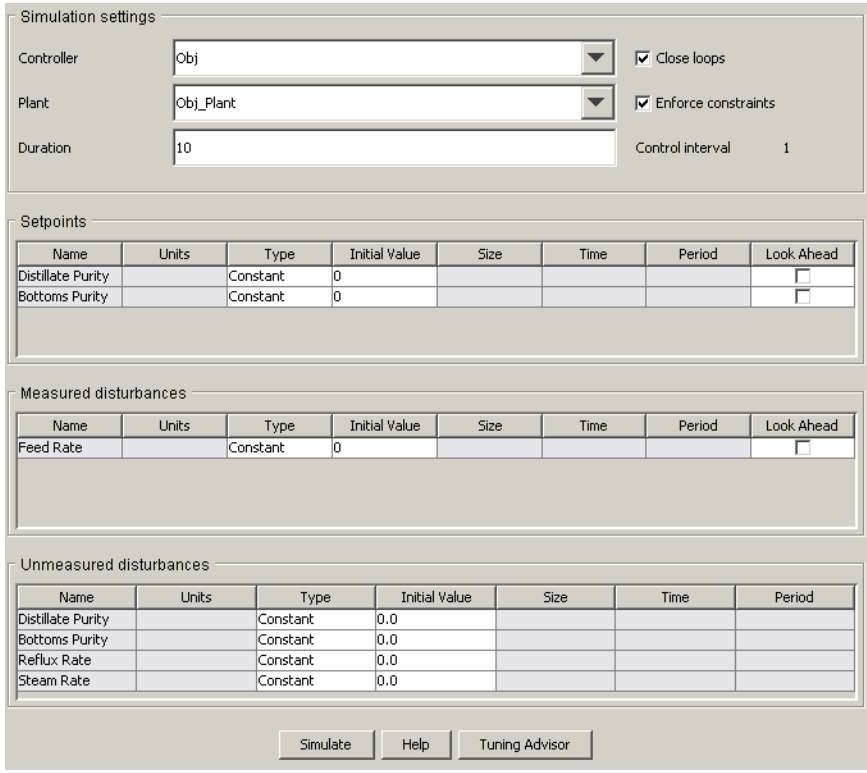
Note Each controller in a design project/task must have a unique name.

Export Controller

Opens the MPC Controller Exporter dialog box (see “Exporting a Controller” on page 8-15).

Simulation Scenario View

This view appears whenever you select one of your scenario specification nodes (see “Tree View” on page 8-6). 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.



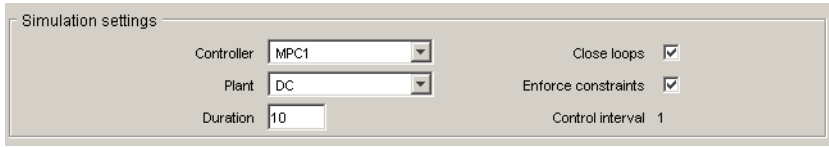
The middle table won't appear unless you have designated at least one input signal to be a measured disturbance.

The following sections describe the view's main features:

- “Model and Horizons Tab” on page 8-30
- “Simulation Settings” on page 8-51
- “Setpoints” on page 8-51
- “Measured Disturbances” on page 8-52
- “Unmeasured Disturbances” on page 8-53
- “Signal Type Settings” on page 8-54
- “Simulation Button” on page 8-56
- “Tuning Advisor Button” on page 8-56

- "Right-Click Menus" on page 8-56

Simulation Settings



Use this section to set the following:

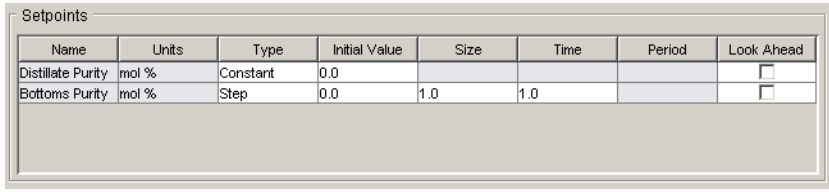
- **Controller** – Select one of your controllers,
- **Plant** – Select the plant model that will act as the "real" plant in the simulation, i.e., it need not be the same as that used for controller predictions.
- **Duration** – The simulation duration in time units.
- **Close loops** – If cleared, the 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 8-30).

Setpoints

Note Setpoint specifications affect *closed-loop* simulations only.

Use this table to specify the setpoint for each output. In the example below, which is for an application having two plant outputs, the first would be constant at 0.0, and the second would change step-wise.



The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. 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.

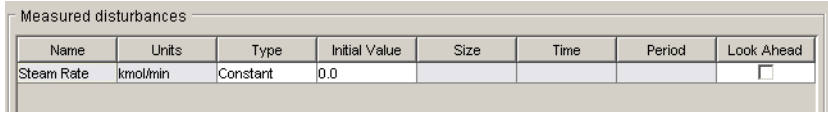
For details on the signal types, see “Signal Type Settings” on page 8-54.

If the **Look Ahead** option is selected (i.e., on), the controller will use future values of the setpoints in its calculations. This improves setpoint tracking, but knowledge of future setpoint changes is unusual in practice.

Note In the current implementation, selecting or clearing the **Look ahead** option for one output will set the others to the same state. Model Predictive Control Toolbox code does not allow you to **Look ahead** for some outputs but not for others.

Measured Disturbances

Use this table to specify the variation of each measured disturbance. In the example below, which is for an application having a single measured disturbance, the “Steam Rate” input would be constant at 0.0.



The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See “Signal Definition View” on page 8-16. 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.

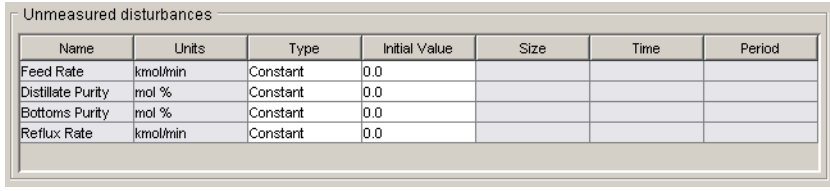
For details on the signal types, see “Signal Type Settings” on page 8-54.

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 disturbances

Name	Units	Type	Initial Value	Size	Time	Period
Feed Rate	kmol/min	Constant	0.0			
Distillate Purity	mol %	Constant	0.0			
Bottoms Purity	mol %	Constant	0.0			
Reflux Rate	kmol/min	Constant	0.0			

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 8-16. 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.

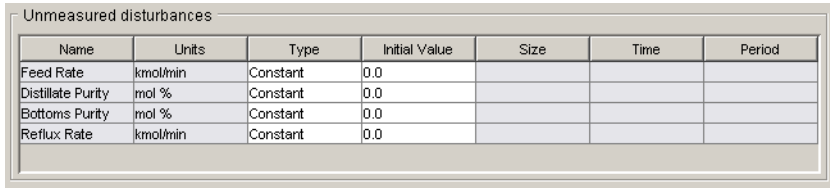
The significance of the **Initial Value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see “Signal Type Settings” on page 8-54.

Open-Loop Simulations

For open-loop simulations, you can vary the MV unmeasured disturbance to simulate the plant's response to a particular MV. The MV signal coming from the controller stays at its nominal value, and the MV unmeasured disturbance adds to it.

For example, suppose `Reflux Rate` is an MV, and the corresponding row in the table below represents an unmeasured disturbance in this MV.


Unmeasured disturbances

Name	Units	Type	Initial Value	Size	Time	Period
Feed Rate	kmol/min	Constant	0.0			
Distillate Purity	mol %	Constant	0.0			
Bottoms Purity	mol %	Constant	0.0			
Reflux Rate	kmol/min	Constant	0.0			

You could set it to a constant value of 1 to simulate the plant's open-loop unit-step response to the `Reflux Rate` input. (In a closed-loop simulation, controller adjustments would also contribute, changing the response.)

Similarly, an unmeasured disturbance in an MO adds to the output signal coming from the plant. If there are no changes at the plant input, the plant outputs are constant, and you see only the change due to the disturbance. This allows you to check the disturbance character before running a closed-loop simulation.

Signal Type Settings

The table below is an example that uses five of the six available signal types (the **Constant** option has been illustrated above). The cells with white backgrounds are the entries you must supply. All have defaults.

Name	Units	Type	Initial Value	Size	Time	Period
Distillate Purity	mol %	Step	0.0	1.0	1.0	
Bottoms Purity	mol %	Ramp	0.0	1.0	1.0	
Reflux Rate	kmol/min	Sine	0.0	1.0	0.0	1.0
Steam Rate	kmol/min	Pulse	0.0	1.0	0.0	1.0
Feed Rate	kmol/min	Gaussian	0.0	1.0	1.0	

Constant

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

$y = y_0$ for $t \geq 0$.

Step

Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal changes step-wise by **Size** units. Its value thereafter = **Initial Value** + **Size**.

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M$ for $t > 0$, where M = **Size**.

Ramp

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

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M(t - t_0)$ for $t \geq t_0$, where M = **Size**.

Sine

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

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

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

Pulse

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

$y = y_0$ for $0 \leq t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M$ for $t_0 \leq t \leq t_0$, where M = **Size**, T = **Period**

$y = y_0$ for $t \geq t_0 + T$

Gaussian

Prior to **Time**, the signal = **Initial Value**. At **Time**, the signal begins to vary randomly about **Initial Value** with standard deviation **Size**.

$y = y_0$ for $0 \le t < t_0$, where y_0 = **Initial Value**, t_0 = **Time**

$y = y_0 + M$randn for $t \ge t_0$, where M = **Size**.

`randn` is the MATLAB random-normal function, which generates random numbers having zero mean and unit variance.

Simulation Button

Click the **Simulate** button to simulate the scenario. You can also press **Ctrl+R**, use the toolbar icon (see "Toolbar" on page 8-6), or use the **MPC/Simulate** menu option (see "Menu Bar" on page 8-4).

Tuning Advisor Button

Click **Tuning Advisor** to open a window that helps you improve your controller's performance. See "Weight Sensitivity Analysis" on page 8-57.

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.

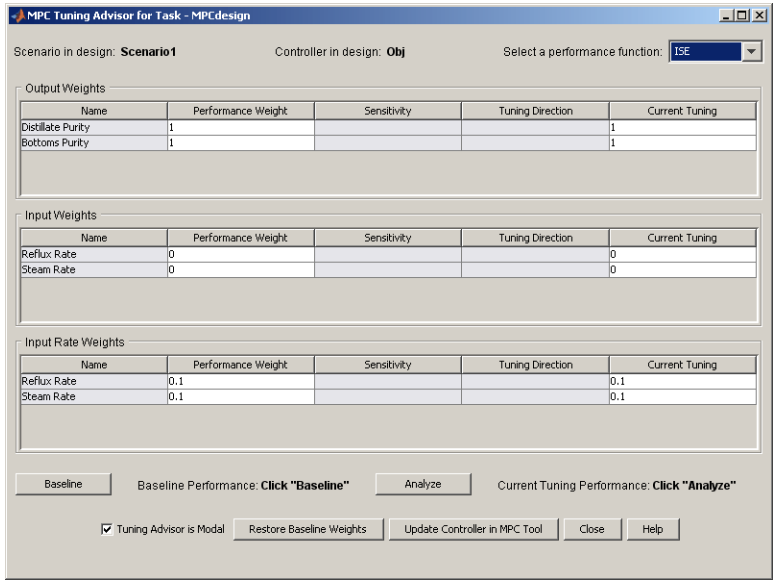
Note Each scenario in a design project/task must have a unique name.

Weight Sensitivity Analysis

In this section...

When you design MPC controllers, you can use the Tuning Advisor to help you determine which weight has the most influence on the closed-loop performance. The Tuning Advisor also helps you determine in which direction to change the weight to improve performance. Using the Advisor, you can know numerically how each weight impacts the closed-loop performance, which makes designing MPC controllers easier when the closed-loop responses does not depend intuitively on the weights.

To start the Tuning Advisor, click **Tuning Advisor** in a simulation scenario view (see "Tuning Advisor Button" on page 8-56). The next figure shows the default Tuning Advisor window for a distillation process in which there are two controlled outputs, two manipulated variables, and one measured disturbance (which the Tuning Advisor ignores). In this case, the originating scenario is **Scenario1**.



The Tuning Advisor populates the **Current Tuning** column with the most recent tuning weights of the controller displayed in the **Controller in Design**. In this case, `Obj` is the controller. The Advisor also initializes the **Performance Weight** column to the same values. The **Scenario in Design** displays the scenario from which you started the Tuning Advisor. The Advisor uses this scenario to evaluate the controller's performance.

The columns highlighted in grey are Tuning Advisor displays and are read-only. For example, signal names come from the “Signal Definition View” on page 8-16 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 metric from the **Select a performance function** drop-down list in the upper right-hand corner of the Advisor. You can choose one of four standard ways to compute the performance measure, *J*. In each case, the goal is to minimize *J*.

- ISE (Integral of Squared Error, the default). This is the standard linear quadratic weighting of setpoint tracking errors, manipulated variable movements, and deviations of manipulated variables from targets (if any). The formula is

$$J = \sum_{i=1}^{Tstop} \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

  where `Tstop` is the number of controller sampling intervals in the scenario, e_{yij} is the deviation of output j from its setpoint (reference) at time step i, e_{uij} is the deviation of manipulated variable j from its target at time step i, Δu_{ij} is the change in manipulated variable j at time step i (i.e., $\Delta u_{ij} = u_{ij} - u_{i-1,j}$), and w_j^y, w_j^u, and $w_j^{\Delta u}$ are nonnegative *performance weights*.
- IAE (Integral of Absolute Error). Similar to the ISE but with squared terms replaced by absolute values

$$J = \sum_{i=1}^{Tstop} \left(\sum_{j=1}^{n_y} | w_j^y e_{yij} | + \sum_{j=1}^{n_u} (| w_j^u e_{uij} | + | w_j^{\Delta u} \Delta u_{ij} |) \right)$$

  The IAE gives less emphasis to any large deviations.
- ITSE (time-weighted Integral of Squared Errors)

$$J = \sum_{i=1}^{Tstop} i\Delta t \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

which penalizes deviations at long times more heavily than the ISE, i.e., it favors controllers that rapidly eliminate steady-state offset.

- ITAE (time-weighted Integral of Absolute Errors)

$$J = \sum_{i=1}^{Tstop} i\Delta t \left(\sum_{j=1}^{n_y} | w_j^y e_{yij} | + \sum_{j=1}^{n_u} (| w_j^u e_{uij} | + | w_j^{\Delta u} \Delta u_{ij} |) \right)$$

which is like the ITSE but with less emphasis on large deviations.

Specify Performance Weights

Each of the above formulae use the same three performance weights, w_j^y, w_j^u, and $w_j^{\Delta u}$. All must be non-negative real numbers. Use the weights to:

- Eliminate a term by setting its weight to zero. For example, a manipulated variable rarely has a target value, in which case you should set its w_j^u to zero. Similarly if a plant output is monitored but doesn't have a setpoint, set its w_j^y to zero.
- Scale the variables so their absolute or squared errors influence J appropriately. For example, an e_{yij} of 0.01 in one output might be as important as a value of 100 in another. If you have chosen the ISE, the first should have a weight of 100 and the second 0.01. In other words, scale all equally important expected errors to be of order unity.

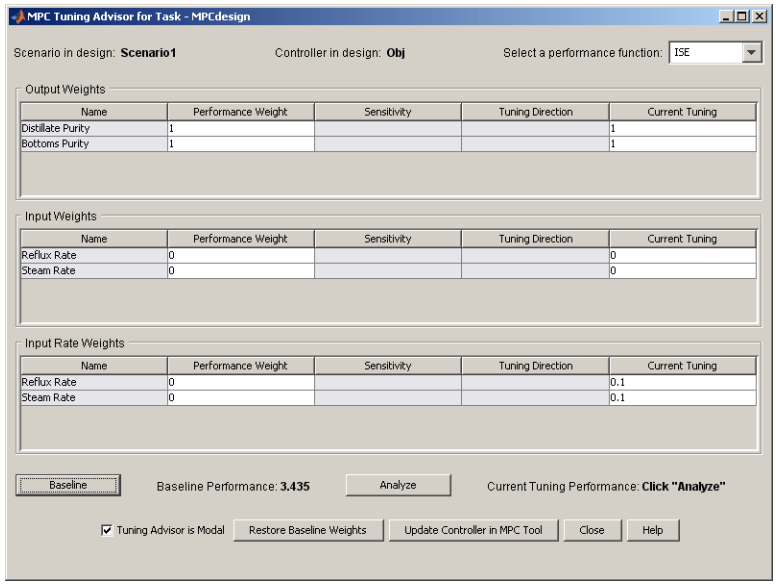
  A Model Predictive Controller uses weights internally as tuning devices. Although there is some common ground, the performance weights and tuning weights should differ in most cases. Choose performance weights to define good performance and then tune the controller weights to achieve it. The Tuning Advisor's main purpose is to make this task easier.

Baseline Performance

After you define the performance metric and specify the performance weights, compute a baseline J for the scenario by clicking **Baseline**. The next figure shows how this transforms the above example (the two $w_j^{\Delta u}$ performance weights have also been set

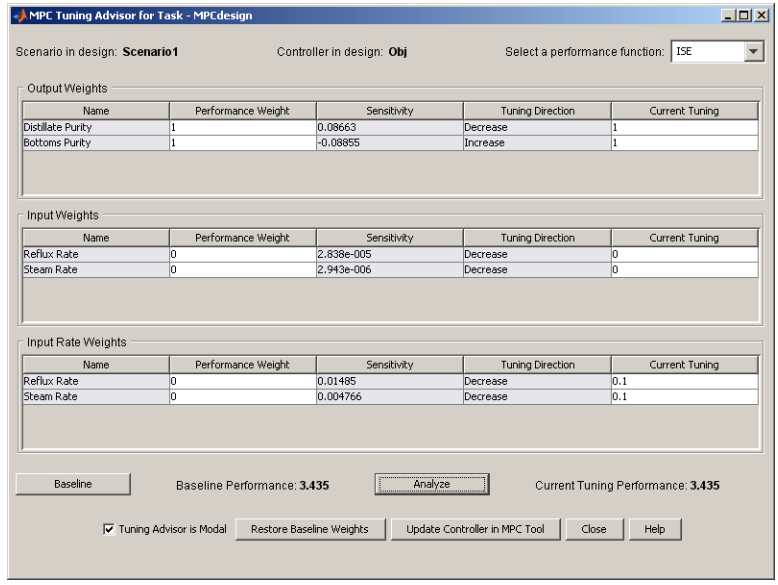
to zero because manipulated variable changes are acceptable if needed to achieve good setpoint tracking for the two (equally weighted) outputs. The computed $J = 3.435$ is displayed in **Baseline Performance**, to the right of the **Baseline** button.

The Tuning Advisor also displays response plots for the scenario with the baseline controller (not shown but discussed in "Customize Response Plots" on page 8-69).



Sensitivities and Tuning Advice

Click **Analyze** 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.

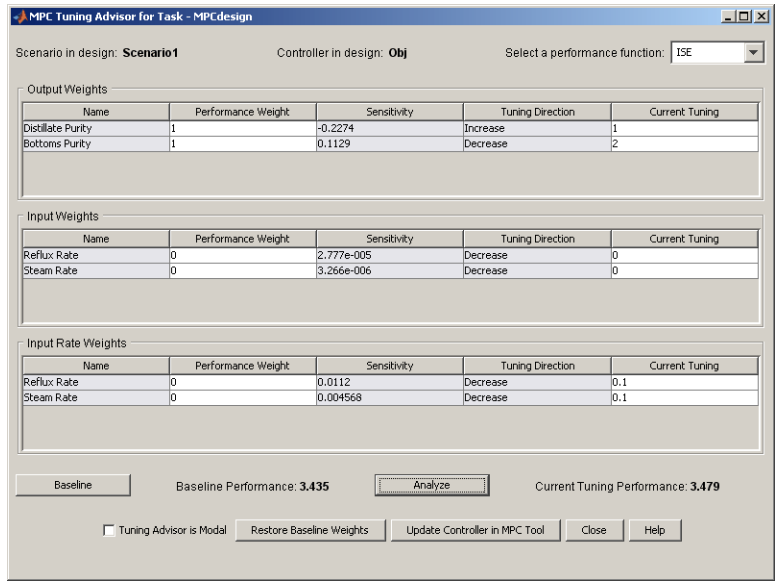
It's best to focus on a small subset of the tuning weights for which the sensitivities suggest good possibilities for improvement.

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^y are the only tuning weights worth considering. Again, it seems unlikely that a change will help much. The display below shows the effect of doubling the tuning weight on the bottoms purity (second) output. Note the 2 in the last column of this row. After you click **Analyze,** the response plots (not shown) make it clear that this output tracks its setpoint more accurately but at the expense of the other, and the overall *J* actually increases.

Notice also that the sensitivities have been recomputed with respect to the revised controller tuning weights. Again, there are no obvious opportunities for improved performance.

Thus, we have quickly determined that the default controller tuning weights are near-optimal in this case, and further tuning is not worth the effort.



Refine Controller Tuning Weights

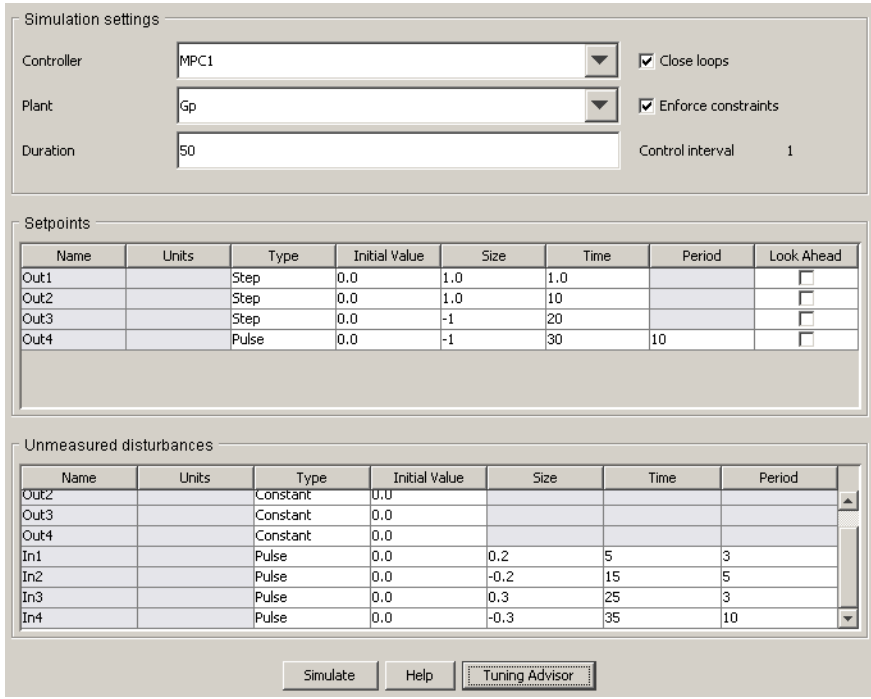
The Tuning Advisor can help you to refine controller tuning weights for better performance. It also provides a quantitative performance measurement.

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.

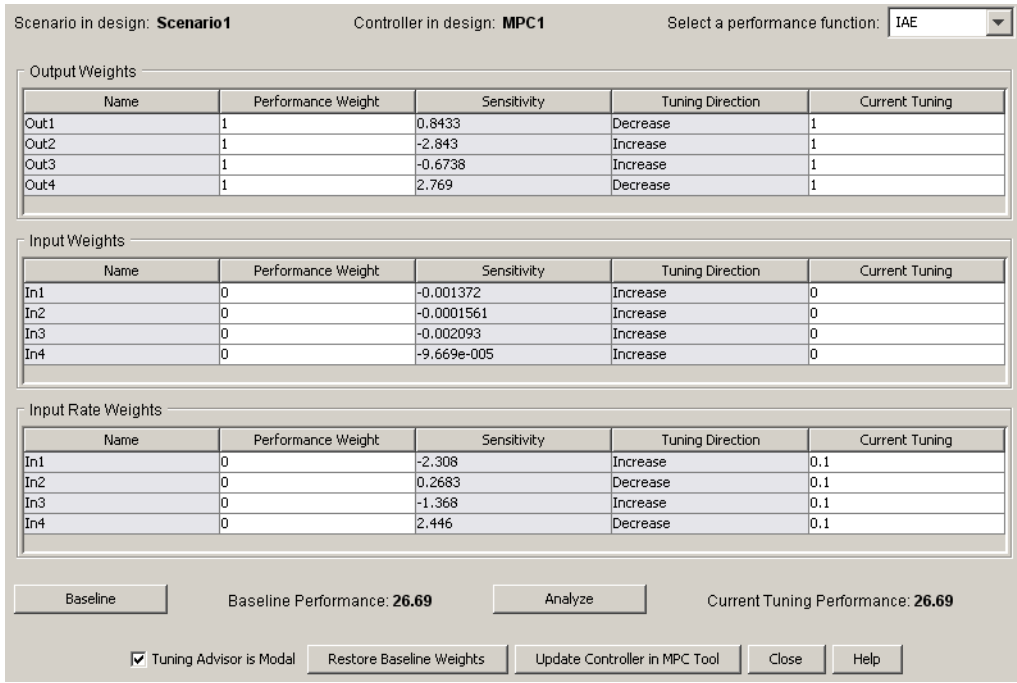
After starting the design tool and importing the plant model, G, which becomes the controller design basis, we accept the default values for all controller parameters. We also load a second plant model, G_p, in which all parameters of G have been perturbed randomly with a standard deviation of 5%.



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**.

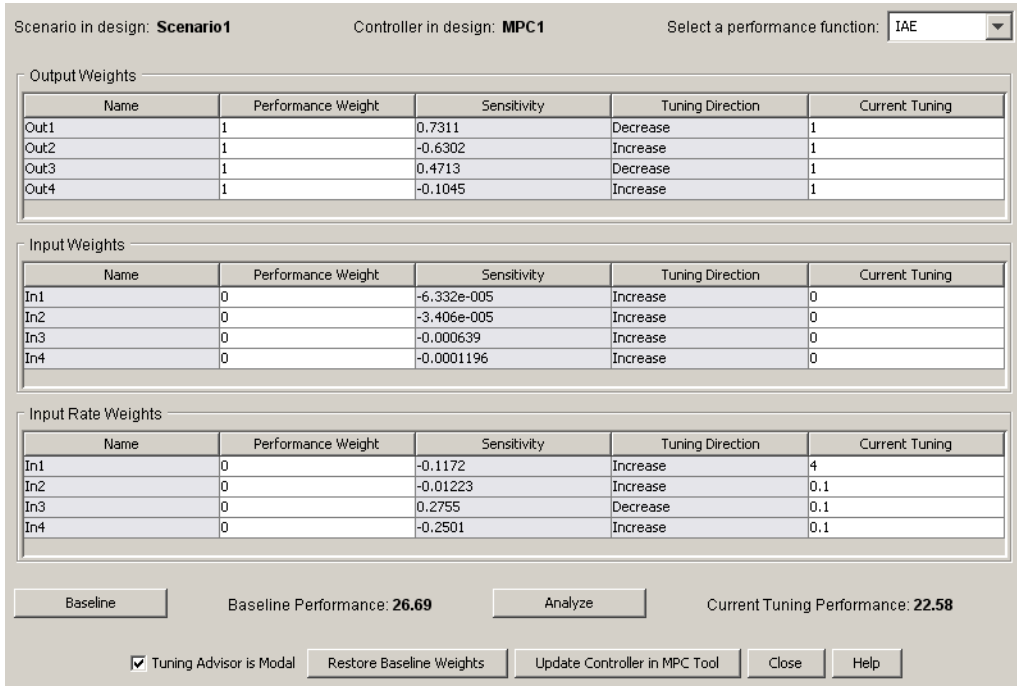


The Tuning Advisor resembles the previous figure. The sensitivity values indicate that a decrease in the `Out4` weight or an increase in the `Out2` weight would have the most

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.

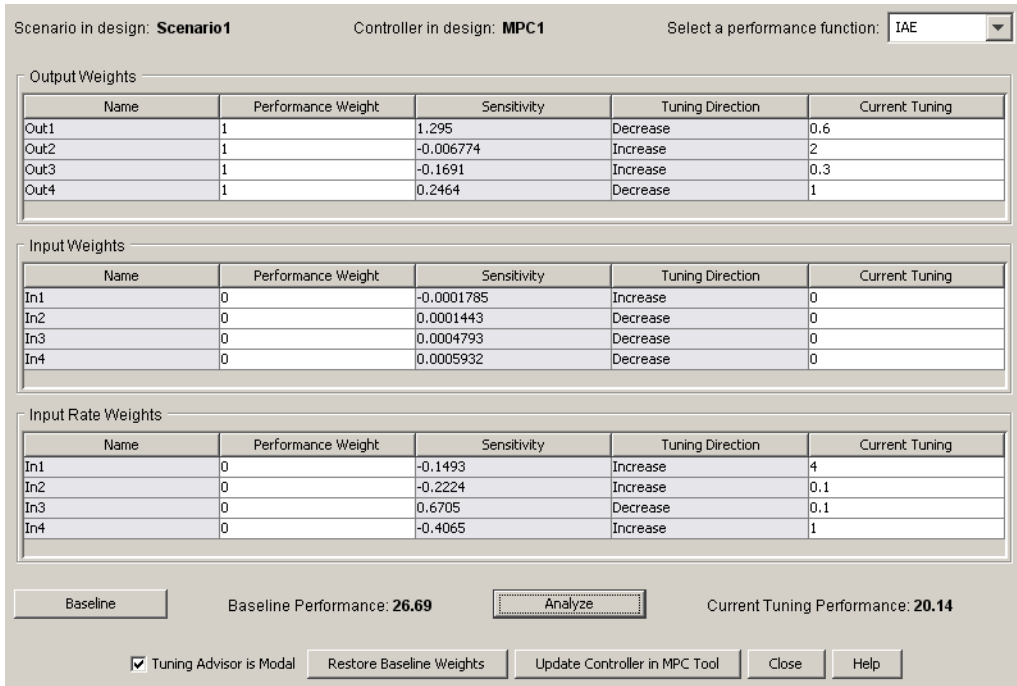
Sensitivities for **Input Rate Weights** `In1` and `In4` are of roughly equal magnitude but the `In4` suggestion is a decrease and this weight is already near its lower bound of zero. Thus, we focus on the `In1` weight.

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 in input rate tuning weights will have little impact.



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%.

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

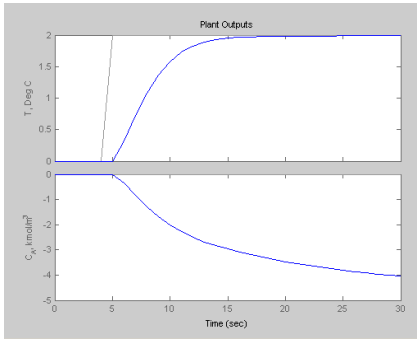
By default, the Advisor window is modal, meaning that you won't be able to access any other MATLAB windows while the Advisor is active. You can disable **Tuning Advisor is Modal**, as shown in the above example. This is *not recommended*, however. In particular, if you return to the Design Tool and modify your controller, your changes won't be communicated to the Advisor. Instead, close the Advisor, modify the controller and then reopen the Advisor.

Scenarios for Performance Measurement

The scenario used with the Advisor should be a true test of controller performance. It should include a sequence of typical setpoint changes and disturbances. It is also good practice to test controller robustness with respect to prediction model error. The usual approach is to define a scenario in which the plant being controlled differs from the controller's prediction model.

Customize Response Plots

Each time you simulate a scenario, the design tool plots the corresponding plant input and output responses. The graphic below shows such a *response plot* for a plant having two outputs (the corresponding input response plot is not shown).



By default, each plant signal plots in its own graph area (as shown above). If the simulation is closed loop, each output signal plot include the corresponding setpoint.

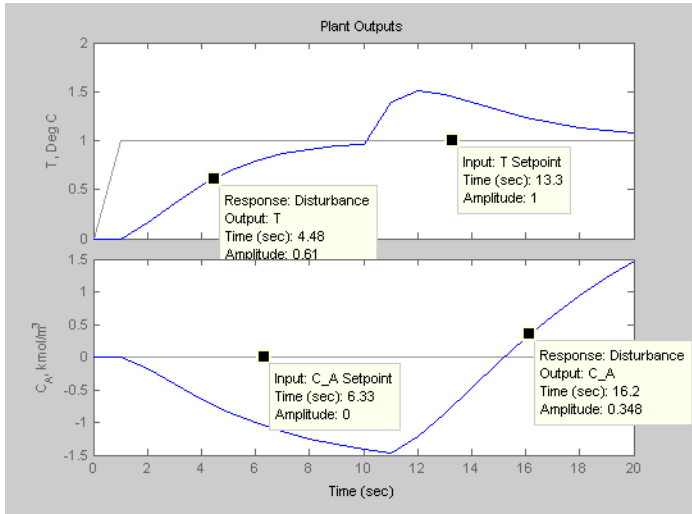
The following sections describe response plot customization options:

Data Markers

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

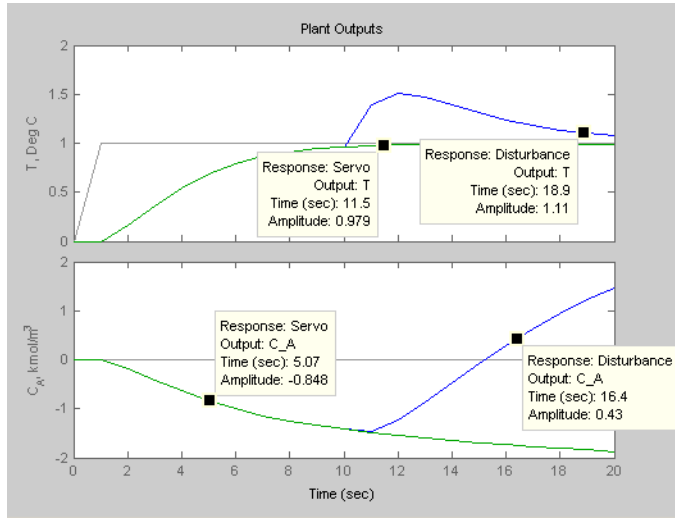
By default the response plots include all the scenarios you've simulated. The example below shows a response plot for a plant with two outputs. The data markers indicate the two scenarios being plotted: “Accurate Model” and “Perturbed Model”. Both scenarios use the same setpoints (not marked—the lighter 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

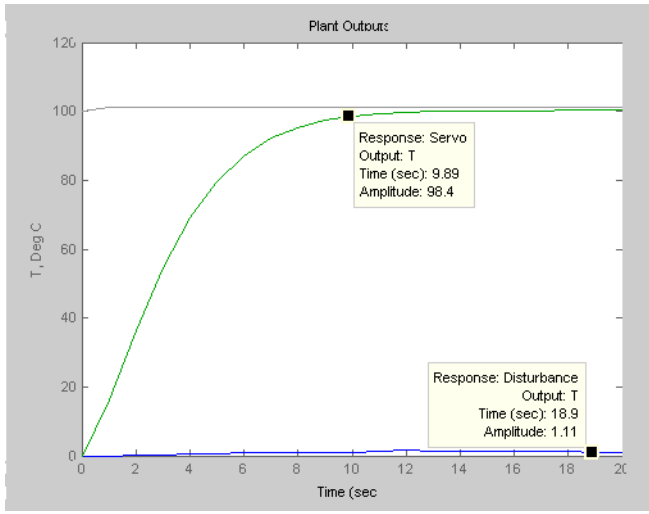
By default, each variable appears in its own plot area. You can instead choose to display variables together in a single plot. To do so, right-click in a plot's white space, and select **Channel Grouping**, and then select **All**.

To return to the default mode, use the **Channel Grouping**: **None** option.

Normalizing Response Amplitudes

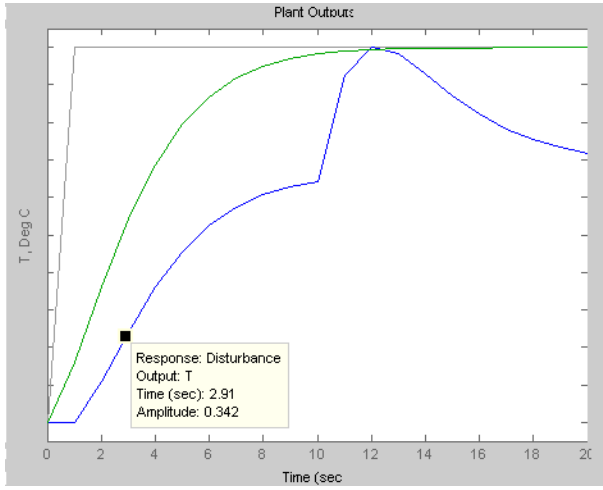
When you're using the **Channel Grouping**: **All** option, you might find that the variables have very different scales, making it difficult to view them together. You can choose to *normalize* the curves, so that each expands or contracts to fill the available plot area.

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 Marker” on page 8-69). 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.


Plant Output
T, Deg C
Response: Disturbance
Output: T
Time (sec): 2.91
Amplitude: 0.342
0
2
4
6
8
10
12
14
16
18
Time (sec