# Model Predictive Control Toolbox™ Release Notes

MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

Trademarks

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

Patents

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

Contents

R2021b

R2021a

R2020b

R2017a

R2016b

R2016a

R2014a

R2013b

R2008a

No New Features or Changes

R2007b

R2007a

Bug Fixes

R2006b

No New Features or Changes

R2006a

R14SP3

No New Features or Changes

R14SP2

No New Features or Changes

R2023a

Version: 8.1

New Features

Bug Fixes

Automatic Differentiation: Speed up multistage nonlinear MPC by generating analytical Jacobian functions

You can now automatically generate Jacobian functions for multistage nonlinear MPC using automatic differentiation. For more information, see `generateJacobianFunction`.

Nonlinear MPC: Generate code for fixed-size arrays

Nonlinear MPC blocks now support fixed-size arrays when generating C/C++ code. Since fixed size arrays can use statically allocated memory, this feature allows you to deploy generated MPC code on embedded systems that do not support dynamic memory allocation. For more information, see "Code Generation for Variable-Size Arrays" (Simulink) and "Generate Code and Deploy Controller to Real-Time Targets". For more information on nonlinear MPC blocks, see Nonlinear MPC Controller and Multistage Nonlinear MPC Controller.

Multistage Nonlinear MPC: Multistage Nonlinear MPC block now supports the ISO 26262 standard

In R2023a, compile-time warnings and errors related to ISO® 26262 have been eliminated for the Multistage Nonlinear MPC Controller Simulink® block. Also, for the same block, and all major ISO 26262 compliance checkpoints in Model Advisor have been cleared.

For more information on see ISO 26262 Support in MATLAB and Simulink.

Finite Control Set MPC: Discrete control set MPC renamed to finite control set MPC

The discrete control set MPC feature is now referred to as finite control set MPC. For more information, see "Finite Control Set MPC".

R2022b

Version: 8.0

New Features

Bug Fixes

Nonlinear MPC: Use neural network as prediction model

You can now use a neural network state space object from System Identification Toolbox™ to implement a prediction model for a nonlinear MPC controller.

To do so, use `generateModelFunctions` to generate the state and Jacobian functions from an `idNeuralStateSpace` object, then set up these functions as a prediction model for your nonlinear MPC controller.

For more information, see `idNeuralStateSpace` (System Identification Toolbox) and `nlssest` (System Identification Toolbox). For more information on nonlinear MPC, see Nonlinear MPC.

Nonlinear MPC: Passivity-based constraints

You can now guarantee closed loop nominal stability for nonlinear MPC controllers by enforcing passivity constraints.

To do so, in the `Passivity` property of your nonlinear MPC controller, set the `EnforceConstraint` field to `true`, and set the `InputFcn` and `OutputFcn` fields to the names of the passivity input and output constraint functions, respectively. You can also set the input and output passivity index using the `OutputPassivityIndex` and `InputPassivityIndex` fields.

For more information, see Specify Constraints for Generic Nonlinear MPC and the `Passivity` property of the `nlmpc` object. For examples about passivity, see Control of Quadruple-tank Using Passivity-Based Nonlinear MPC and Control of Robot Manipulator Using Passivity-Based Nonlinear MPC.

Vehicle Path Planner System Block: Design and implement vehicle path planner using nonlinear MPC

Using the new Vehicle Path Planner System block, you can design and implement a multistage nonlinear MPC controller to plan a collision-free path from an initial pose to a target pose. You can use this block to plan paths for a vehicle, such as a vehicle entering and exiting a parking spot or a robot moving in a warehouse.

Once you have planned a path, you can have the vehicle track the path using a path-following control system.

R2022a

Version: 7.3

New Features

Bug Fixes

MISRA C 2012: Implement MISRA C:2012-compliant controllers with linear MPC and ADAS blocks in Simulink

In R2022a, compile-time warnings and errors related to MISRA C™:2012 have been eliminated for the following blocks.

- MPC Controller
- Adaptive MPC Controller
- Multiple MPC Controllers
- Adaptive Cruise Control System
- Lane Keeping Assist System
- Path Following Control System

Also, all key mandatory and required rules of MISRA C:2012 for Automated-Generated Code compliance defined in Polyspace® Bug Finder™ have been cleared for these blocks. For more information on Polyspace Bug Finder with MISRA C see Required or Mandatory MISRA C:2012 Rules Supported by Polyspace Bug Finder.

For more information on MISRA C:2012 compliance, see MISRA C:2012 Compliance Considerations (Simulink).

Explicit MPC: Explicit MPC blocks now support both the ISO 26262 and MISRA C:2012 standards

In R2022a, compile-time warnings and errors related to MISRA C:2012 and ISO 26262 have been eliminated for the following blocks:

- Explicit MPC Controller
- Multiple Explicit MPC Controllers

Also, for these blocks, all major ISO 26262 compliance checkpoints in Model Advisor as well as all key mandatory and required rules of MISRA C: 2012 for Automated-Generated Code compliance defined in Polyspace Bug Finder have been cleared.

For more information on running Model Advisor checks see Check Your Model Using the Model Advisor (Simulink). For more information on Polyspace Bug Finder with MISRA C see Required or Mandatory MISRA C:2012 Rules Supported by Polyspace Bug Finder.

For more information on compliance with both standards, see ISO 26262 Support in MATLAB and Simulink and MISRA C:2012 Compliance Considerations (Simulink).

Nonlinear MPC: Use controller blocks inside Triggered and Function Call subsystems

You can now use the Nonlinear MPC Controller and Multistage Nonlinear MPC Controller blocks inside a Triggered Subsystem (Simulink) or a Function-Call Subsystem (Simulink). To do so, you must specify the sample time of the nonlinear MPC block as -1, so it can inherit the sample time of its parent subsystem.

Simulate an MPC controller faster using buildMex

You can now speed up command line simulations of linear MPC controllers using the `buildMEX` command to generate a MEX file that you can later use instead of `mpcmove` or `mpcmoveCodeGeneration`.

New Nonlinear MPC Examples: Implement controllers for parallel parking and passivity-based control

The following new reference examples demonstrate applications of nonlinear MPC.

- Parallel Parking of Truck-Trailer Using Multistage Nonlinear MPC — Parallel park a truck-trailer system using multistage nonlinear MPC.
- Control of Robot Manipulator Using Passivity-Based Nonlinear MPC — Enforce stability in a robotic-manipulator control system by adding passivity-based constraints to a nonlinear MPC controller.

R2021b

Version: 7.2

New Features

Bug Fixes

Compatibility Considerations

Implement Multistage Nonlinear MPC Controllers with Embotech FORCESPRO Solver

You can now simulate and generate code for nonlinear multistage MPC controllers using the NLP solver shipped with the FORCESPRO real-time embedded optimization software, developed by Embotech AG. FORCESPRO provides a Model Predictive Control Toolbox plugin that allows you to generate and deploy custom solvers that are optimized for your specific MPC problems, thereby achieving a superior computational performance.

For more information, see Multistage Nonlinear MPC and Implement MPC Controllers using Embotech FORCESPRO Solvers.

Custom QP Solvers: Use quadprog from Optimization Toolbox as an active-set custom QP solver

You can now use the QP active-set solver available with `quadprog` (Optimization Toolbox) as an alternative QP solver to the built-in algorithms for simulation and code generation in linear MPC applications.

To set the `mpcobj` MPC object to use the `quadprog` function as a custom QP solver, at the MATLAB® command prompt, enter the following.

`setCustomSolver(mpcobj,'quadprog')`

Here, `mpcobj` is an `mpc` object.

The command generates two MATLAB functions, `mpcCustomSolver.m` and `mpcCustomSolverCodeGen.m`, and saves them in your current folder. The command also updates `mpcobj` to let it use these functions, which wrap `quadprog`. You can also further customize these functions, for example by adjusting the solver options, provided that you use the active-set solver (since other `quadprog` solvers are not supported for MPC problems).

For more information, see `setCustomSolver`, QP Solvers and `active-set quadprog Algorithm` (Optimization Toolbox).

ISO 26262: Implement ISO 26262-compliant controllers with linear MPC and ADAS blocks in Simulink

In R2021b, compile-time warnings and errors related to ISO 26262 have been eliminated for the following blocks.

- MPC Controller
- Adaptive MPC Controller
- Multiple MPC Controllers
- Adaptive Cruise Control System
- Lane Keeping Assist System
- Path Following Control System

Also, all major ISO 26262 compliance checkpoints in Model Advisor have been cleared for these blocks. For more information on running Model Advisor checks see Check Your Model Using the Model Advisor (Simulink).

For more information, see ISO 26262 Support in MATLAB and Simulink.

Functionality being removed or changed

Support for opening MPC Design Tool sessions saved before release R2015b has been removed
Errors

Support for opening MPC Design Tool sessions saved before release R2015b has been removed.

If you have sessions saved before release R2015b, open and resave the session files using **MPC Designer** in any release from R2015b through R2021a.

mpctool command has been removed
Errors

The `mpctool` command has been removed. Instead, to open **MPC Designer**, enter `mpcDesigner` at the MATLAB command prompt.

R2021a

Version: 7.1

New Features

Bug Fixes

Compatibility Considerations

Discrete Control Set MPC: Solve linear MPC problems with manipulated variables belonging to discrete sets

You can now solve linear MPC problems in which some or all manipulated variables belong to discrete sets. To do so, for a given manipulated variable, specify the new `Type` field of the corresponding `ManipulatedVariables` structure in the `mpc` object:

- `'binary'` — Restrict the manipulated variable to be either 0 or 1.
- `'integer'` — Restrict the manipulated variable to be an integer.
- Vector containing a discrete set of admissible values — Restrict the manipulated variable to the specified values, for example `mpcobj.MV(2).Type=[-1,0,0.5,1,2];`.

By default, the type is set to `'continuous'`, indicating that the manipulated variable is continuous.

You can simulate a discrete set linear MPC controller in:

- MATLAB — using `sim`, `mpcmove`, `mpcmoveAdaptive` and `mcmoveMultiple`.
- Simulink — using the MPC Controller, Adaptive MPC Controller, and Multiple MPC Controllers blocks.

When simulating multiple controllers using `mcmoveMultiple` or the Multiple MPC Controllers block, all candidate controllers must use the same manipulated variable type configuration.

Code generation from a controller with discrete control sets is supported in both MATLAB and Simulink.

A new internal mixed-integer quadratic programming (MIQP) solver is used to solve the discrete set MPC problem. You can use the new property `Optimizer.MixedIntegerOptions` of the `mpc` object to customize the options for this solver (like for example number of iterations and constraints tolerance).

For more information and examples on discrete control set MPC see Discrete Control Set MPC, Solve a Discrete Set MPC Problem in MATLAB , and Surge Tank Control Using Discrete Control Set MPC.

Multistage Nonlinear MPC: Solve nonlinear MPC problems efficiently with multiple stage cost and constraints

You can now solve nonlinear MPC problems in which different cost and constraint functions are defined for different specific prediction steps (stages). Specifically, a multistage MPC controller with a prediction horizon of length p has $p+1$ stages (where the first stage is the one active at the current time, and the last stage is the terminal one).

For a multistage MPC controller, each stage has its own decision variables and parameter vector, as well as its own nonlinear cost and constraints (which are only functions of the decision variables and parameter vector at this stage). This allows for a much more efficient computation and storage of the gradient functions, thereby resulting in general in much faster execution times.

To create a multistage nonlinear MPC object, use `nlmpcMultistage`, and define the cost and constraint functions for each stage. You can additionally specify standard bounds on inputs and states.

You can simulate a multistage MPC controller in:

- MATLAB — using `nlmpcmove`.
- Simulink — using the Multistage Nonlinear MPC Controller block.

Compared to nonlinear MPC, neither plant output nor weight are explicitly defined in `nlmpcMultistage` objects, since you can weight any combination of state and input in each stage cost and constraint function. Additionally, you can specify number of slack variables needed at any stage.

Code generation from a nonlinear multistage controller is supported in both MATLAB and Simulink.

For examples on how to create and use multistage MPC controller, see Create and Simulate a Multistage Nonlinear MPC Controller, Simulate a Multistage Nonlinear MPC Controller Using Initial Guesses and Truck and Trailer Automatic Parking Using Multistage Nonlinear MPC.

GPU Code Generation Support in MATLAB and Simulink

From both MATLAB and Simulink, you can now generate code for linear MPC controllers that use a GPU.

GPU Code Generation in MATLAB

To generate and use GPU code in MATLAB:

1 Create data structures from your linear MPC controller using the `getCodeGenerationData` function.

2 Optionally simulate your controller using the `mpcmoveCodeGeneration` function and the data structures, instead of using the `mpcmove` function.

3 Create a coder configuration option object using the `coder.gpuConfig` function, and configure the code generation options.

4 Generate code for the `mpcmoveCodeGeneration` function using the `codegen` function and the coder configuration options object. Doing so generates a new function that uses code running on the GPU.

5 Simulate your controller using the generated function and the data structures.

For more information, see Generate Code and Deploy Controller to Real-Time Targets.

For an example on using GPU code in MATLAB see Use the GPU to Compute MPC Moves in MATLAB.

Code Generation in Simulink

You can generate and use GPU code from the MPC Controller, Adaptive MPC Controller, or Explicit MPC Controller blocks.

To generate GPU code from a Simulink model containing any of these blocks, open the Configuration Parameters dialog box by clicking **Model Settings**. Then, in the **Code Generation** section, select **Generate GPU code**.

For more information, see Generate Code and Deploy Controller to Real-Time Targets.

For details on how to configure your model for GPU code generation, see Code Generation from Simulink Models with GPU Coder (GPU Coder).

Functionality being removed or changed

Support for opening MPC Design Tool sessions saved before release R2015b will be removed
Warns

Support for opening MPC Design Tool sessions saved before release R2015b will be removed in release R2021b.

If you have sessions saved before release R2015b, open and resave the session files using **MPC Designer** in any release from R2015b through R2021a.

R2020b

Version: 7.0

New Features

Bug Fixes

Compatibility Considerations

Implement MPC Controllers using Embotech FORCESPRO Solvers

You can use a Model Predictive Control Toolbox plugin developed by Embotech AG to simulate and generate code for linear and nonlinear MPC controllers. The plugin leverages the FORCESPRO real-time embedded optimization software, and allows you to generate custom solvers that are highly optimized on your specific MPC problem and deploy them on real-time hardware, achieving a superior computational performance.

For more information, see Implement MPC Controllers using Embotech FORCESPRO Solvers.

Online Time-Varying Constraints: Simulate MPC controllers with constraints that vary over the prediction horizon at run time

You can now simulate any MPC controller with constraints that vary over the prediction horizon at run time. You can do so both at the command line and in Simulink. This method of specifying constraints is the same as that already used for nonlinear MPC controllers.

To vary constraints at the command line, specify the `OutputMin`, `OutputMax`, `MVMin`, and `MVMax` properties of an `mpcmoveopt` object as arrays. Each row in the array contains the constraints for one prediction horizon step. You can then use these constraints when simulating a controller using `mpcmove`, `mpcmoveAdaptive`, or `mpcmoveMultiple`. For more information on the format of these arrays, see the corresponding properties of `nlmpcmoveopt`.

To vary constraints in Simulink, connect matrix signals to the `ymin`, `ymax`, `umin`, and `umax` constraint input ports of the MPC Controller, Adaptive MPC Controller, or Multiple MPC Controllers blocks. Each row in the matrix signal contains the constraints for one prediction horizon step. For more information on the format of these matrix signals, see the `y.min`, `y.max`, `mv.min`, and `mv.max` signals for the Nonlinear MPC Controller block.

Compatibility Considerations

To specify an input or output constraint that varies across the prediction horizon, you set the corresponding property of the MPC controller object as a vector. Each element of the vector contains the constraint value for one prediction horizon step.

In previous releases, at run time, you could change only the first element of this constraint vector. The controller retained the constraint profile by adding a constant offset to the remaining elements.

If your application requires maintaining the constraint profile across the prediction horizon, you can still use the previous method for specifying constraints.

If your application requires changing the constraint profile at run time, you can use the new method for specifying constraints.

Reference Examples: Design and simulate automated driving applications that use model predictive controllers

The following new examples show how to automated driving applications that use model predictive controllers.

- Highway Lane Following with Intelligent Vehicles

- Traffic Light Negotiation with Unreal Engine Visualization

R2020a

Version: 6.4

New Features

Bug Fixes

Compatibility Considerations

Interior-Point QP Solver: Efficiently compute optimal control moves for large-scale MPC problems

You can now design an MPC controller that uses an interior-point QP solver instead of the default active-set solver. Using an interior-point solver can provide superior performance for large-scale optimization problems, such as MPC applications that enforce constraints over large prediction and control horizons. For more information on configuring the solver for your MPC controller, see QP Solvers.

You can also use the new interior-point solver to develop your own custom model predictive controllers or as a general-purpose QP solver that supports code generation. To do so, use the `mpcInteriorPointSolver` function. To specify solver options, use the `mpcInteriorPointOptions` function.

Compatibility Considerations

The method for configuring QP solver options for an MPC controller object has changed. For more information, see "MPC controller properties for configuring active-set QP solver have changed" on page 7-3.

To distinguish the existing active-set QP solver from the new interior-point QP solver, the `mpcqpsolver` and `mpcqpsolverOptions` functions are replaced by the new `mpcActiveSetSolver` and `mpcActiveSetOptions` functions, respectively. For more information, see "mpcqpsolver will be removed" on page 7-4 and "mpcqpsolverOptions will be removed" on page 7-3.

Nonlinear MPC Code Generation: Generate code for nonlinear MPC controllers that use default fmincon solver with the SQP algorithm

You can now generate code for nonlinear MPC controllers that use the default `fmincon` solver with the SQP algorithm. You can generate code in both MATLAB and Simulink.

Code Generation in MATLAB

To generate code in MATLAB:

1 Create data structures from your nonlinear MPC controller using the `getCodeGenerationData` function.
2 Simulate your controller using the `nlmpcmoveCodeGeneration` function in place of `nlmpcmove`.
3 Generate code for the `nlmpcmoveCodeGeneration` function.

Code Generation in Simulink

You generate code for the Nonlinear MPC Controller block in the same manner as you do for the other MPC controller Simulink blocks.

If your nonlinear MPC controller uses optional parameters, you connect the output of a Bus Creator block to the `params` input port of the block. To generate code for such a controller, you must place the Nonlinear MPC Controller block and Bus block together in a subsystem. Then, you can generate code for the subsystem.

Simulate Nonlinear MPC Controller Using MEX File

You can speed up simulation of nonlinear MPC controllers by simulating your controller using a MEX file instead of calling `nlmpcmove`. Doing so is useful when you run multiple simulations using the same nonlinear MPC controller design. You can also use the generated MEX file when simulating your controller using the Nonlinear MPC Controller block.

To create the MEX file, use the new `buildMEX` function.

Reference Examples: Design model predictive controllers for automated driving applications

The following new examples show how to design and simulate model predictive controllers for automated driving applications.

- Parallel Parking Using Nonlinear Model Predictive Control
- Parallel Parking Using RRT Planner and MPC Tracking Controller
- Parking Valet Using Nonlinear Model Predictive Control
- Highway Lane Change
- Traffic Light Negotiation
- Automate Testing for Highway Lane Following

Functionality Being Removed or Changed

MPC controller properties for configuring active-set QP solver have changed
Warns

The MPC controller properties for configuring the default active-set QP solver have changed. These changes require updates to your code.

Settings specific to the active-set solver are now in the `Optimizer.ActiveSetOptions` property of the controller instead of the `Optimizer` property. Any settings that apply to both the default active-set solver and the new interior-point solver remain in the `Optimizer` controller property.

Update Code

This table shows the MPC controller QP solver properties that have changed and how to update your code.

Not Recommended	Recommended
`mpcobj.Optimizer.MaxIter`	`mpcobj.Optimizer.ActiveSetOptions.MaxIterations`
`mpcobj.Optimizer.UseWarmStart`	`mpcobj.Optimizer.ActiveSetOptions.UseWarmStart`

mpcqpsolverOptions will be removed
Warns

`mpcqpsolverOptions` will be removed in a future release. Use `mpcActiveSetOptions` instead. There are differences between these functions that require updates to your code.

Update Code

To update your code:

- Change the function name from `mpcqpsolverOptions` to `mpcActiveSetOptions`. The syntaxes are equivalent.
- Some field names of the returned structure have changed. The default field values are the same. This table shows the new property names.

Previous Property Name	New Property Name
MaxIter	MaxIterations
FeasibilityTol	ConstraintTolerance

- The returned structure of `mpcActiveSetOptions` contains the new field `UseHessianAsInput`. To continue to use the inverse of the lower-triangular decomposition of the Hessian matrix with `mpcActiveSetSolver`, you must set `UseHessianAsInput` to `false`.

mpcqpsolver will be removed
Warns

`mpcqpsolver` will be removed in a future release. Use `mpcActiveSetSolver` instead. There are differences between these functions that require updates to your code.

Update Code

The following differences require updates to your code:

- For `mpcActiveSetSolver`, you define inequality constraints in the form $Ax \leq b$. Previously, for `mpcqpsolver`, you defined inequality constraints in the form $Ax \geq b$.
- For `mpcActiveSetSolver`, you specify solver options with a structure created using the `mpcActiveSetOptions` function. Previously, for `mpcqpsolver`, you created an option structure using the `mpcqpsolverOptions` function. These option structures contain the same options, though some option names have changed.
- By default, you pass the Hessian matrix to `mpcActiveSetSolver`. Previously, you passed the inverse of lower-triangular Cholesky decomposition (`Linv`) of the Hessian matrix to `mpcqpsolver`. To continue to use `Linv`, set the `UseHessianAsInput` field of the structure returned by `mpcActiveSetSolver` to `false`.
- When your QP problem has either no inequality constraints or no equality constraints, the corresponding `A` or `Aeq` input argument to `mpcActiveSetSolver` must be `zeros(0,n)`, where `n` is the number of decision variables. Previously, for `mpcqpsolver`, you specified these input arguments as `[]`.

This table shows some typical usages of `mpcqpsolver` and how to update your code to use `mpcActiveSetSolver` instead.

Not Recommended	Recommended
```opt = mpcqpsolverOptions;` `[x,status] = mpcqpsolver(Linv,f,A,b,...` ` Aeq,beq,iA0,opt);```	```opt = mpcActiveSetOptions;` `opt.UseHessianAsInput = false;` `[x,status] = mpcActiveSetSolver(Linv,f,...` ` -A,-b,Aeq,beq,iA0,opt);``` Alternatively, you can use the Hessian matrix H. ```opt = mpcActiveSetOptions;` `[x,status] = mpcActiveSetSolver(H,f,...` ` -A,-b,Aeq,beq,iA0,opt);```

Not Recommended	Recommended

`opt = mpcqpsolverOptions('single');`
`[x,status] = mpcqpsolver(Linv,f,A,b,...`
`    Aeq,beq,iA0,opt);`
	`opt = mpcActiveSetOptions('single');`
	`opt.UseHessianAsInput = false;`
	`[x,status] = mpcActiveSetSolver(Linv,f,...`
	`        -A,-b,Aeq,beq,iA0,opt);`

`opt = mpcqpsolverOptions;`
`opt.MaxIter = 300;`
`opt.FeasibilityTol = 1e-5;`
`[x,status] = mpcqpsolver(Linv,f,A,b,...`
`    Aeq,beq,iA0,opt);`
	`opt = mpcActiveSetOptions;`
	`opt.UseHessianAsInput = false;`
	`opt.MaxIterations = 300;`
	`opt.ContraintTolerance = 1e-5;`
	`[x,status] = mpcActiveSetSolver(Linv,f,...`
	`        -A,-b,Aeq,beq,iA0,opt);`

`[x,status] = mpcqpsolver(Linv,f,[],...`
`    zeros(0,1),Aeq,beq,iA0,opt);`
	`n = length(f);`
	`opt.UseHessianAsInput = false;`
	`[x,status] = mpcActiveSetSolver(Linv,f,...`
	`        zeros(0,n),zeros(0,1),Aeq,beq,iA0,opt);`

`[x,status] = mpcqpsolver(Linv,f,A,b,...`
`    [],zeros(0,1),iA0,opt);`
	`n = length(f);`
	`opt.UseHessianAsInput = false;`
	`[x,status] = mpcActiveSetSolver(Linv,f,...`
	`        -A,-b,zeros(0,n),zeros(0,1),iA0,opt);`

mpchelp has been removed

mpchelp has been removed. To view Model Predictive Control Toolbox help at the command line, type help mpc. To view the Model Predictive Control Toolbox documentation, type doc mpc.

R2019b

Version: 6.3.1

Bug Fixes

Reference Examples: Design model predictive controllers for robotics and automated driving applications

The following new examples show how to design and simulate model predictive controllers for robotics and automated driving:

- Control of Quadrotor Using Nonlinear Model Predictive Control
- Lane Change Assist Using Nonlinear Model Predictive Control
- Plan and Execute Collision-Free Trajectories using KINOVA Gen3 Manipulator (Robotics System Toolbox)

R2019a

Version: 6.3

New Features

Bug Fixes

Path Following Control System Block: Design, simulate, and implement lane-following controllers in Simulink

You can now design, simulate, and implement model predictive controllers for path following using the new Path Following Control System block. This block combines the functionality of the Adaptive Cruise Control System and Lane Keeping Assist System blocks. You can design your controller either with or without safe distance spacing control.

You can generate a custom subsystem for this block, which you can modify for your application. This option is useful when you want to:

- Modify default MPC settings or use advanced MPC features.
- Modify the default controller initial conditions.
- Use different application settings, such as a custom safe following distance definition for adaptive cruise control.

Run-Time Horizon Tuning: Make run-time updates to controller prediction and control horizons

You can now vary the prediction and control horizons of MPC and adaptive MPC controllers at run time.

To vary the horizons at run time from the command line, at each control interval, specify the `PredictionHorizon` and `ControlHorizon` properties of the `mpcmoveopt` object. You can then pass the `mpcmoveopt` object to either `mpcmove` or `mpcmoveAdaptive`.

In Simulink, to vary the horizons for an MPC Controller or Adaptive MPC Controller block, select the **Adjust prediction horizon and control horizon at run time** parameter. Doing so adds p and m input ports to the block for the prediction and control horizons, respectively. You must specify the maximum prediction horizon using the **Maximum prediction horizon** parameter.

Online horizon tuning supports code generation in both MATLAB and Simulink. To generate code for a controller that uses online horizon tuning, you must enable dynamic memory allocation.

For more information, see Adjust Horizons at Run Time.

Nonlinear MPC: Simulate as an adaptive or linear time-varying MPC controller to determine if a linear controller provides comparable performance

In practice, when producing comparable performance, linear MPC is preferred over nonlinear MPC due to its higher computational efficiency. You can now determine whether a linear controller provides comparable performance to the nonlinear case by simulating your nonlinear MPC controller as either an adaptive or linear time-varying MPC controller.

To do so, set the `Optimization.RunAsLinearMPC` property your `nlmpc` controller object to one of the following:

- `"Adaptive"` — For each control interval, a linear model is obtained from the specified nonlinear state and output functions at the current operating point and used across the prediction horizon.

To determine if an adaptive MPC controller provides comparable performance to the nonlinear controller, use this option. For more information on adaptive MPC, see Adaptive MPC.

- "TimeVarying" — For each control interval, *p* linear models are obtained from the specified nonlinear state and output functions at the *p* operating points predicted from the previous interval, one for each prediction horizon step. To determine if a linear time-varying MPC controller provides comparable performance to the nonlinear controller, use this option. For more information on time-varying MPC, see Time-Varying MPC.

To use either the "Adaptive" or "TimeVarying" option, your controller must have no custom constraints and no custom cost function.

Nonlinear MPC: Linearly interpolate block moves when using manipulated variable blocking with nonlinear MPC controllers

By default, nonlinear MPC controllers use piecewise constant blocking intervals when you specify the control horizon as a vector, which is often too restrictive for optimal path planning applications. Therefore, previously, you would specify your control horizon to be approximately equal to your prediction horizon, which can produce a poorly conditioned nonlinear programming problem.

To produce a less-restrictive, better-conditioned nonlinear programming problem, you can now specify piecewise linear manipulated variable blocking intervals. To do so, set the Optimization.MVInterpolationOrder property of your nlmpc controller object to 1.

For more information, see Manipulated Variable Blocking.

R2018b

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

Nonlinear MPC: Design and simulate model predictive controllers with nonlinear prediction models, constraints, and cost functions

You can now design and simulate model predictive controllers with nonlinear prediction models, constraints, and cost functions using the new `nlmpc` object and Nonlinear MPC Controller Simulink block. You can use nonlinear MPC controllers for:

- Closed-loop control of a nonlinear plant
- Optimal trajectory planning

By default, nonlinear MPC controllers use `fmincon` as their nonlinear solver, which requires Optimization Toolbox™ software. For more information on nonlinear model predictive control, see Nonlinear MPC.

Run-Time Tuning of Time-Varying Weights: Make run-time updates to controller weights that change over the prediction horizon

You can now make run-time updates to controller tuning weights that change over the prediction horizon when simulating implicit MPC controllers, adaptive MPC controllers, and gain-scheduled MPC controllers at the command line and in Simulink.

To specify time-varying weights at the command line, at each control interval create an `mpcmoveopt` object, and set the following weight properties using an array:

- `OutputWeights` — Output variable tuning weights
- `MVWeights` — Manipulated variable tuning weights
- `MVRateWeights` — Manipulated variable rate tuning weights

You can use these specified weights when simulating a controller using `mpcmove`, `mpcmoveAdaptive`, or `mpcmoveMultiple`.

To specify time-varying weights in Simulink, enable the following input ports on the MPC Controller, Adaptive MPC Controller, or Multiple MPC Controllers blocks, and connect a matrix signal:

- `y.wt` — Output variable tuning weights
- `u.wt` — Manipulated variable tuning weights
- `du.wt` — Manipulated variable rate tuning weights

MPC Simulink Block Optimal Sequences: Obtain optimal predicted outputs and states

You can now obtain the predicted optimal sequences for output and state variables from the following Simulink blocks:

- MPC Controller
- Adaptive MPC Controller
- Multiple MPC Controllers

To access the optimal sequences, add the following output ports to your block:

- `y.seq` — Optimal output variable sequence
- `x.seq` — Optimal state variable sequence

For more information, see the block reference pages.

Compatibility Considerations

The dimensions of the optimal control sequence output port, `mv.seq`, have changed. For more information, see "MPC Simulink block mv.seq output port signal dimensions have changed" on page 10-4.

MPC Simulink Block Run-Time Constraints: Independently enable input and output constraints

You can now independently enable the ports for only the input and output constraints that you want to vary at run time when using the following Simulink blocks:

- MPC Controller
- Adaptive MPC Controller
- Multiple MPC Controllers

You can independently enable the following run-time constraint ports:

- `umin` — Lower manipulated variable bounds
- `umax` — Upper manipulated variable bounds
- `ymin` — Lower output variable bounds
- `ymax` — Upper output variable bounds

For more information, see the block reference pages.

Lane Keeping Assist System Block: Model transport lag in ego car dynamic model

You can now model transport lag in your ego vehicle model when using the Lane Keeping Assist System block. This lag can include actuator, sensor, and communication lags.

review Function Results Structure: Programmatically review controller designs and obtain test results

The `review` function now returns a structure that contains test result flags. When you return the tests results, the `review` function suppresses the HTML testing report.

Using this results structure, you can now programmatically review the designs of one or more MPC controllers.

ADAS Examples: Design controllers for lane following

The following new examples show how to design and simulate model predictive controllers for lane following:

- Lane Following Control with Sensor Fusion and Lane Detection
- Lane-Following Control with Monocular Camera Perception

Functionality Being Removed or Changed

Support for implementing economic MPC using a linear MPC controller has been removed
Errors

Support for implementing economic MPC using a linear MPC controller has been removed. Implement economic MPC using the new nonlinear MPC controller instead. For more information on nonlinear MPC controllers, see Nonlinear MPC.

Update Code

If you previously saved a linear MPC object configured with custom cost or constraint functions, the software generates a warning when the object is loaded and an error if it is simulated. To suppress the error and warning messages and continue using your linear MPC controller, `mpcobj`, without the custom costs and constraints, set the `IsEconomicMPC` flag to `false`.

```
mpcobj.IsEconomicMPC = false;
```

To implement your economic MPC controller using a nonlinear MPC object:

1 Create an `nlmpc` object.
2 Convert your custom cost function to the format required for nonlinear MPC. For more information on nonlinear MPC cost functions, see Specify Cost Function for Nonlinear MPC.
3 Convert your custom constraint function to the format required for nonlinear MPC. For more information on nonlinear MPC constraints, see Specify Constraints for Nonlinear MPC.
4 Implement your linear prediction model using state and output functions. For more information on nonlinear MPC prediction models, see Specify Prediction Model for Nonlinear MPC.

MPC Simulink block mv.seq output port signal dimensions have changed
Behavior change

The signal dimensions of the `mv.seq` output port of the MPC Controller, Adaptive MPC Controller, and Multiple MPC Controllers blocks have changed. Previously, this signal was a p-by-N_{mv} matrix, where p is the prediction horizon and N_{mv} is the number of manipulated variables. Now, `mv.seq` is a $(p+1)$-by-N_{mv} matrix, where row $p+1$ duplicates row p.

R2018a

Version: 6.1

New Features

Bug Fixes

ADAS Blocks: Design, simulate, and implement adaptive cruise control and lane-keeping algorithms

You can now design, simulate, and implement model predictive controllers for automatic cruise control and lane keeping assistance using new Simulink blocks. These blocks provide simplified application-specific interfaces for configuring model predictive controllers.

For both blocks, you can generate a custom subsystem, which you can modify for your application. This option is useful when you want to:

- Modify default MPC settings or use advanced MPC features.
- Modify the default controller initial conditions.
- Use different application settings, such as a custom safe following distance definition for adaptive cruise control.

For more information, see Adaptive Cruise Control System and Lane Keeping Assist System.

R2017b

Version: 6.0

New Features

Bug Fixes

Economic MPC: Design and simulate model predictive controllers with arbitrary nonlinear cost function and constraints

Model Predictive Control Toolbox software now supports economic MPC; that is, the ability to optimize the controller for an arbitrary cost function under arbitrary nonlinear constraints. Previously, MPC controllers supported only quadratic cost functions with linear constraints. Unlike nonlinear MPC, economic MPC uses linear prediction models and linear state estimation.

Economic MPC requires Optimization Toolbox software.

For more information, see Economic MPC.

Fast MPC: Guarantee worst-case execution time by using approximate QP solution

You can now guarantee the worst-case execution time for your MPC controller by applying an approximate (suboptimal) solution after the number of optimization iterations exceeds a specified maximum value. This feature applies when using the built-in QP solver, a custom QP solver, and `fmincon` for economic MPC.

For more information, see Suboptimal QP Solution.

Custom QP Solvers: Generate code for third-party QP solvers written in C/C++ or MATLAB code suitable for code generation

You can now generate code for MPC controllers that use a custom QP solver written in either C/C++ or MATLAB code that is suitable for code generation. The controller calls this solver in place of the built-in QP solver at each control interval. For an example, see Simulate and Generate Code for MPC Controller with Custom QP Solver.

For more information on:

- Custom QP solvers, see Custom QP Solver.
- Code generation, see Generate Code and Deploy Controller to Real-Time Targets.

Generating code for MPC controllers with custom QP solvers:

- At the command line requires MATLAB Coder™ software.
- In Simulink requires Simulink Coder or Simulink PLC Coder™ software.

Mixed Input/Output Constraints: Update constraints on linear combinations of inputs and outputs at run time

You can now update mixed input/output constraints at run time when simulating your MPC controller at the command line or in Simulink. You can update these constraints for traditional and adaptive MPC controllers. You can also generate code for controllers that use online mixed input/output constraints.

For an example that uses online custom constraints with an adaptive MPC controller, see Obstacle Avoidance Using Adaptive Model Predictive Control. For more information on updating constraints at run time, see Update Constraints at Run Time.

ADAS Examples: Design controllers for adaptive cruise control, autonomous vehicle steering, and obstacle avoidance

New examples show how to design and simulate advanced driver assistance systems (ADAS) using model predictive controllers. The new examples are:

- Adaptive Cruise Control System Using Model Predictive Control
- Autonomous Vehicle Steering Using Model Predictive Control
- Obstacle Avoidance Using Adaptive Model Predictive Control
- Adaptive Cruise Control with Sensor Fusion

R2017a

Version: 5.2.2

New Features

Bug Fixes

New Examples: Design and simulate nonlinear model predictive controller in MATLAB and Simulink

Two new examples show how to design and simulate a nonlinear model predictive controller in MATLAB and Simulink. Like linear MPC, nonlinear MPC also solves a constrained optimization problem at each control interval. However, since the plant model is nonlinear, the nonlinear MPC converts the optimal control problem into a nonlinear optimization problem with a nonlinear cost function and nonlinear constraints. The new examples are:

- Swing-up Control of a Pendulum Using Nonlinear Model Predictive Control
- Nonlinear MPC Control of an Ethylene Oxidation Plant

Both examples require Optimization Toolbox software.

R2016b

Version: 5.2.1

New Features

Bug Fixes

Compatibility Considerations

Multiple Explicit MPC Controllers Block: Implement gain-scheduled explicit MPC controllers in Simulink

You can now use the Multiple Explicit MPC Controllers block to implement gain-scheduled explicit MPC controllers in Simulink. You design an explicit MPC controller for each operating point and specify a switching signal to choose the active controller. During each control interval, the active explicit MPC controller determines control actions using a table-lookup control law. The inactive controllers continue estimating plant states, which allows for bumpless transfer when switching controllers. The Multiple Explicit MPC Controllers block reduces online computational effort compared to the Multiple MPC Controllers block.

For more information, see Multiple Explicit MPC Controllers.

MPC Designer App: Automatically estimate controller sample time when setting internal plant using stable, continuous-time model

To facilitate controller design, **MPC Designer** now automatically estimates the initial controller sample time when you specify a stable continuous-time plant prediction model. In this case, the app sets the controller sample time to $0.1T_r$, where T_r is the average rise time of the plant. Previously, **MPC Designer** set the controller sample time to a default value of 1.

If you specify an unstable continuous-time plant, **MPC Designer** sets the controller sample time to 1 by default. If you specify a discrete-time plant, **MPC Designer** sets the controller sample time to the plant sample time.

For more information on specifying the MPC controller internal model, see MPC Designer and Design Controller Using MPC Designer.

MPC Designer App: Load previously saved session when opening app from command line

You can now load a previously saved session when opening the **MPC Designer** app. The saved session data includes all plants, controllers, and scenarios in the **Data Browser**, the current MPC structure, and the current plot configuration.

To open **MPC Designer** and load the session saved in savedSession.mat, type the following at the MATLAB command line.

```
mpcDesigner('savedSession.mat')
```

For more information, see MPC Designer.

MPC Designer App: Import identified linear models

You can now import linear System Identification Toolbox models, such as idss or idtf systems, directly into the MPC Designer app.

MPC Designer converts the specified model to a state-space (ss) system and creates a default MPC controller using the state-space system as the internal prediction model. The app discards any noise channels in the identified model.

For more information on designing MPC controllers for identified plants, see Design MPC Controller for Identified Plant Model.

MPC Controller Creation Using Identified Linear Models: Discard noise channels by default

When you create an MPC controller using a linear System Identification Toolbox model with noise channels, the software now discards any noise channels from the model by default. Previously, the noise channels were converted to unmeasured disturbances by default.

Starting in R2016b, to convert noise channels to unmeasured disturbances, first convert the identified model to a state-space model using the `'augmented'` option.

```
ssModel = ss(idModel,'augmented');
```

This option creates `Measured` and `Noise` input groups in `ssModel`. You can then create an MPC controller using the state-space model.

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

The channels in the `Noise` input group are converted to unmeasured disturbances.

For more information on designing MPC controllers for identified plants, see Design MPC Controller for Identified Plant Model.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations

For System Identification Toolbox models with noise channels, `controller = mpc(idModel,Ts)` now discards noise channels by default.	Still works	`ssModel = ss(idModel,'augmented')`
		
		`controller = mpc(ssModel,Ts)`
			To convert noise channels to unmeasured disturbances, convert the identified model to a state-space model using the `'augmented'` option. For more information, see "MPC Controller Creation Using Identified Linear Models: Discard noise channels by default" on page 14-3.

getmpcdata	Error	• get • getconstraint • getEstimator • getindist • getoutdist	Replace all instances of `getmpcdata` using the command corresponding to your specific data of interest.
setmpcdata	Error	• set • setconstraint • setEstimator • setindist • setoutdist	Replace all instances of `setmpcdata` using the command corresponding to your specific data of interest.

Functionality	Result	Use This Instead	Compatibility Considerations
The fully qualified class name `mpcdata.state` has changed to `mpcstate`.	Not applicable	Not applicable	For `mpcstate` state object `x`: • `class(x)` now returns `'mpcstate'`. • `isa(x,'mpcdata.mpcstate')`, now returns `false`. Use `isa(x,'mpcstate')` instead. Otherwise, there is no change in functionality.

R2016a

Version: 5.2

New Features

Bug Fixes

Adaptive MPC with Time-Varying Prediction Models: Simulate adaptive MPC controllers with prediction models that change over the prediction horizon

You can now specify prediction models and nominal conditions that change over the prediction horizon when using adaptive MPC. Use these options if you can predict how the plant and nominal conditions vary in the future.

To vary the prediction model, specify the `Plant` input argument of `mpcmoveAdaptive` as an array of up to p+1 delay-free, discrete-time, state-space models, where p is the prediction horizon of your MPC controller. To vary the nominal conditions, specify the `Nominal` input argument of `mpcmoveAdaptive` as an array of up to p+1 nominal condition structures.

For more information, see Time-Varying MPC and Time-Varying MPC Control of a Time-Varying Plant.

mpcmoveCodeGeneration Command: Generate C code for computing optimal manipulated variable control moves

You can now generate C code for computing optimal manipulated variable control moves for any valid implicit or explicit MPC controller using the new `mpcmoveCodeGeneration` command.

Use the new `getCodeGenerationData` command to create the input data structures for `mpcmoveCodeGeneration`.

For an example of how to generate C code for computing optimal MPC control moves, see Generate Code To Compute Optimal MPC Moves in MATLAB.

Custom QP Solver: Simulate model predictive controllers with a QP solver of your choice

You can now define a custom QP solver for your MPC controller. To do so, you must provide a custom `mpcCustomSolver.m` file on the MATLAB path that finds an optimal solution to the general form QP problem. For more information on the MPC QP problem and how to specify a custom solver, see Custom QP Solver.

For an example on how to use a custom QP solver, see Simulate MPC Controller with a Custom QP Solver.

R2015b

Version: 5.1

New Features

Bug Fixes

Compatibility Considerations

Redesigned MPC Designer App: Design model predictive controllers in MATLAB and Simulink using improved interactive workflows

The redesigned MPC Designer app streamlines MATLAB and Simulink workflows for designing model predictive controllers. You can now:

- Generate MATLAB scripts for MPC controller design tasks. See "MATLAB Script Generation from MPC Designer App: Automatically script model predictive controller design tasks" on page 16-2.
- Generate a Simulink model with an MPC controller and plant model. See "Simulink Model Generation from MPC Designer App: Automatically create a Simulink model with an MPC controller and plant model" on page 16-2.
- Compare responses for multiple MPC controllers in the same plot. See "Comparison of responses for multiple model predictive controllers in the same plot using MPC Designer app" on page 16-3.
- Review MPC controllers for design and run-time stability issues. See "Review model predictive controller design using MPC Designer app" on page 16-3.
- Tune controller performance objectives using interactive sliders. See "Interactive tuning of model predictive controller performance objectives" on page 16-3.

To open the MPC Designer app, enter the following:

```
mpcDesigner
```

For examples of using the app from MATLAB and Simulink, see Design Controller Using MPC Designer and Design MPC Controller in Simulink.

MATLAB Script Generation from MPC Designer App: Automatically script model predictive controller design tasks

You can now generate MATLAB scripts for creating and simulating model predictive controllers designed in the **MPC Designer** app. Generated MATLAB scripts are useful when you want to programmatically reproduce designs that you obtained interactively.

For more information, see Generate MATLAB Code from MPC Designer.

Simulink Model Generation from MPC Designer App: Automatically create a Simulink model with an MPC controller and plant model

You can now generate a Simulink model that uses the current model predictive controller to control its internal plant model. You can then use the generated model to validate your controller design and generate code for real-time control applications.

For more information, see Generate Simulink Model from MPC Designer.

mpcqpsolver Command: Develop and generate code for custom model predictive controllers using KWIK quadratic programming solver

You can use the new `mpcqpsolver` command to develop custom model predictive controllers. Use the new `mpcqpsolverOptions` command to specify additional solver options.

You can also use `mpcspsolver` as a general purpose QP solver that supports code generation.

For more information, see Solve Custom MPC Quadratic Programming Problem and Generate Code.

Review model predictive controller design using MPC Designer app

You can now review your model predictive controllers for potential run-time stability and numerical problems from within the **MPC Designer** app. To review the design of your current controller, on the

Tuning tab, click **Review Design** .

For more information on reviewing model predictive controller designs, see `review` and Review Model Predictive Controller for Stability and Robustness Issues.

Comparison of responses for multiple model predictive controllers in the same plot using MPC Designer app

You can now simultaneously compare the response plots for multiple model predictive controllers using the **MPC Designer** app. On the **MPC Designer** tab, in the **Compare Controllers** drop-down list, select the controllers to compare.
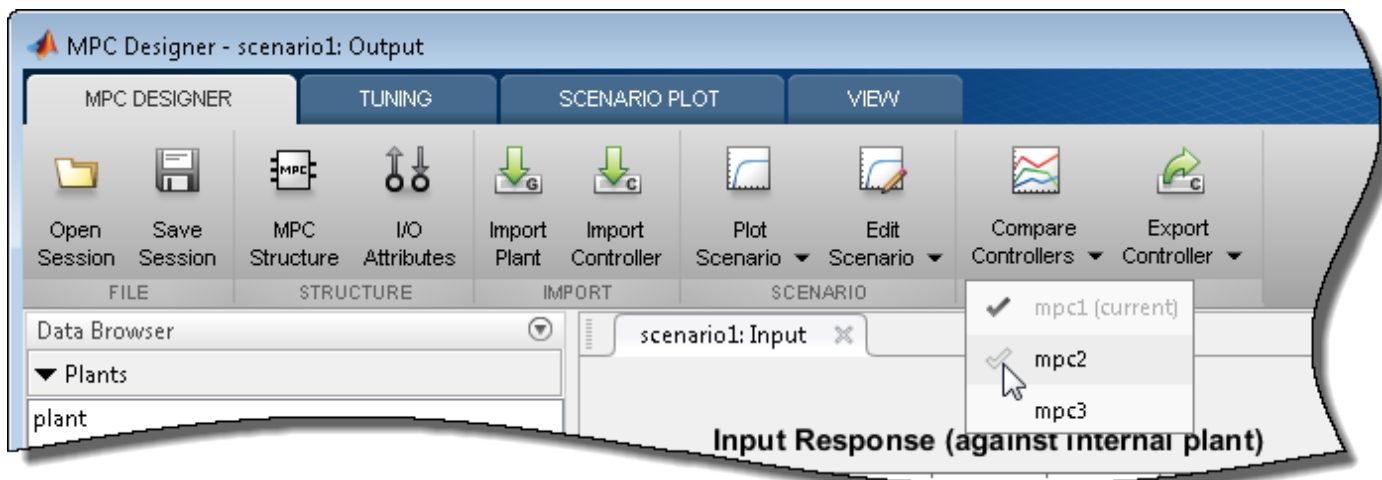
You can add additional controllers to the MPC Designer **Data Browser** by:

-
    Importing a controller from the MATLAB workspace — Select **Import Controller** .
-
    Copying the current controller — Select **Store Controller** .

For more information, see Compare Multiple Controller Responses Using MPC Designer.

Interactive tuning of model predictive controller performance objectives

You can now tune controller performance objectives using interactive sliders. On the **Tuning** tab, use the **Performance Tuning** sliders to adjust the following:

- **Closed-Loop Performance** objective — Moving towards more aggressive control simultaneously increases OV/MV weights and decreases MV Rate weights, which leads to tighter control of outputs and more aggressive control moves. Moving towards more robust control decreases OV/MV weights and increases MV Rate weights, which leads to relaxed control of outputs and more conservative control moves.
- **State Estimation** speed — Moving towards faster state estimation simultaneously increases the gains for disturbance models and decreases the gains for noise models , which leads to more aggressive disturbance rejection. Moving towards slower state estimation decreases the gains for disturbance models and increases the gains for noise models, which leads to more conservative disturbance rejection.
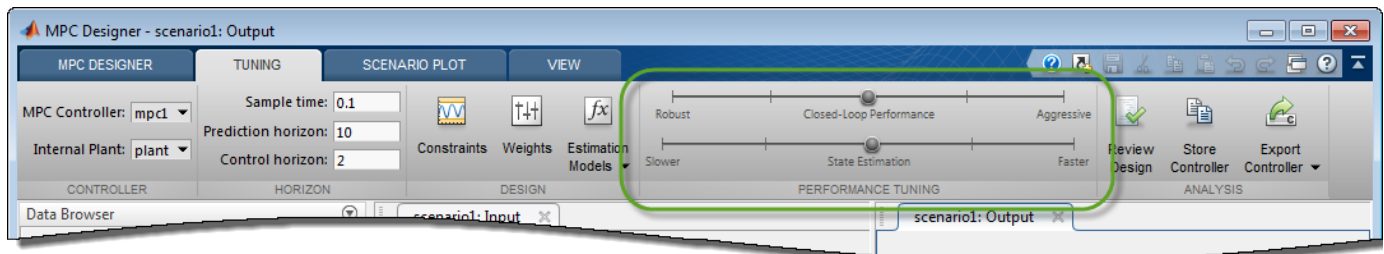
mpctool command renamed to mpcDesigner

The `mpctool` command has been renamed. Starting in R2015b, open the **MPC Designer** app using the new `mpcDesigner` command.

For more information, see MPC Designer.

Compatibility Considerations

If you have scripts or functions that use `mpctool`, consider replacing those calls with `mpcDesigner`.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
mpctool	Warns	mpcDesigner	Consider replacing mpctool with mpcDesigner.

R2015a

Version: 5.0.1

New Features

Bug Fixes

Compatibility Considerations

OutputVariables Integrator property of MPC controller being removed

The MPC controller property `OutputVariables(i).Integrator`, or `OV(i).Integrator`, is being removed. Previously, you specified custom integrator gains in the default output disturbance model using `OV(i).Integrator`. Starting in R2015a, you directly specify a custom output disturbance model as shown:

```
% Define a 2-by-2 plant model with no direct feedthrough
Plant = rss(2,2,2);
Plant.D = 0;
% Create an MPC object
mpcobj = mpc(Plant,1);
% Retrieve the default output disturbance model
Dmodel = getoutdist(mpcobj);
% Change the integrator gains
Dmodel = Dmodel * [2 0;0 3];
% Use new disturbance model in mpcobj
setoutdist(mpcobj,'model',Dmodel)
```

Compatibility Considerations

If your code uses the `OV(i).Integrator` property, you can update your code to use `setoutdist` and `getoutdist` for managing MPC controller output disturbance models.

For example, replace:

```
mpcobj.OV(1).Integrator = 2;
mpcobj.OV(2).Integrator = 3;
```

with:

```
Dmodel = getoutdist(mpcobj);
Dmodel = Dmodel * [2 0;0 3];
setoutdist(mpcobj,'model',Dmodel)
```

Use `tf(getoutdist(mpcobj))` to validate that the results are equivalent.

setoutdist command 'remove' syntax being removed

The `setoutdist(mpcobj,'remove',channels)` syntax is being removed. Previously, you removed integrators from particular channels in the output disturbance model using this syntax. Starting in R2015a, you directly specify a custom output disturbance model as shown:

```
% Define a 2-by-2 plant model with no direct feedthrough
Plant = rss(2,2,2);
Plant.D = 0;
% Create an MPC object
mpcobj = mpc(Plant,1);
% Retrieve the default output disturbance model
Dmodel = getoutdist(mpcobj);
% Remove the output disturbance model from output #1
Dmodel = sminreal([0;Dmodel(2,2)]);
% Use new disturbance model in mpcobj
setoutdist(mpcobj,'model',Dmodel)
```

When removing integrators from output disturbance channels, use `sminreal` to make the custom model structurally minimal.

Compatibility Considerations

If your code uses the `setoutdist(mpcobj,'remove',channels)` syntax, you can update your code to use `setoutdist` and `getoutdist` for managing MPC controller output disturbance models.

For example, replace:

```
setoutdist(mpcobj,'remove',1)
```

with:

```
Dmodel = getoutdist(mpcobj);
Dmodel = sminreal([0;Dmodel(2,2)]);
setoutdist(mpcobj,'model',Dmodel)
```

Use `tf(getoutdist(mpcobj))` to validate that the results are equivalent.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`mpcobj.OV(i).Integrator = value`	Warns	`setoutdist(mpcobj, 'model', sys)`	Use `setoutdist(mpcobj, 'model', sys)` to define custom output disturbance models. For more information, see "OutputVariables Integrator property of MPC controller being removed" on page 17-2
`value = mpcobj.OV(i).Integrator`	Warns	`sys = getoutdist(mpcobj)`	Use `getoutdist(mpcobj)` to retrieve MPC output disturbance models. For more information, see "OutputVariables Integrator property of MPC controller being removed" on page 17-2
`setoutdist(mpcobj, 'remove', channels)`	Warns	`setoutdist(mpcobj, 'model', sys)`	Use `setoutdist(mpcobj, 'model', sys)` to define custom output disturbance models. For more information, see "setoutdist command 'remove' syntax being removed" on page 17-2

R2014b

Version: 5.0

New Features

Bug Fixes

Compatibility Considerations

Explicit MPC control for applications with fast sample times using precomputed solutions

You can now design, simulate and deploy explicit MPC controllers for your plant. This functionality is useful for applications with fast sample times using pre-computed solutions.

To obtain an explicit MPC controller, you must first design a traditional MPC (also called implicit MPC) that is able to achieve your control objectives. Use the `generateExplicitMPC` command to design explicit MPC controllers. Use the `mpcmoveExplicit` command and the Explicit MPC Controller block to simulate explicit MPC controllers at the command-line and in Simulink, respectively.

For more information, see the following 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

Adaptive MPC control through run-time changes to internal plant model

You can now simulate and deploy adaptive MPC controllers for your plant. This functionality helps you control a nonlinear plant across a wide operating range when the new linear plant model is available at run time.

To obtain an adaptive MPC controller, you must first design a traditional MPC (also called implicit MPC) that is able to achieve your control objectives at the initial operating condition. Then, update the internal plant model at each control interval at run time. Use the `mpcmoveAdaptive` command and the Adaptive MPC Controller block to simulate adaptive MPC controllers at the command-line and in Simulink, respectively.

For more information, see the following examples:

- Adaptive MPC Control of Nonlinear Chemical Reactor Using Successive Linearization
- Adaptive MPC Control of Nonlinear Chemical Reactor Using Online Model Estimation

ScaleFactor property for MPC controllers, for making weight tuning independent of the engineering units of input and output variables

You can now specify scale factor in MPC controller in engineering units. The scale factors make weights dimensionless. Choosing proper scale factors, i.e. the operating ranges of the variable, makes weight tuning much easier. The default value of this property is 1.

For more information, see Using Scale Factor to Facilitate Weight Tuning.

Option to use custom state estimation or measured state values instead of the built-in state estimation in MPC controllers

In addition to built-in state estimation, MPC controllers can now run custom state estimation. You can specify the state estimation mode by using `setEstimator(mpcobj,'default')` and `setEstimator(mpcobj,'custom')`, respectively.

When using custom state estimation, you can use the `Plant`, `Disturbance` and `Noise` properties of the controller state object `mpcstate` to provide custom state values at each control interval. The values can be from direct state measurements or your own state estimation algorithm. You must not programmatically change the `LastMV` property in the `mpcstate` object because it is still automatically updated by `mpcmove`.

For more information, see Using Custom State Estimation.

Compatibility Considerations

If your code changes the `LastMV` property of the state object to provide an external MV at run time, you must update the code to use `mpcmoveopt` and specify the value in the `mpcmoveopt.MVused` field instead.

If your code uses the `Plant`, `Disturbance` and `Noise` properties of the state object to provide external state values, you must use `setEstimator(mpcobj,'custom')` to specify the controller to use the custom estimation mode before control starts.

Option to specify manipulated variable target

You can now specify targets on the manipulated variables during run time. At the command line, specify the value in the `MVTarget` field of the `mpcmoveopt` object. In the MPC controller blocks, select **Targets on manipulated variables (mv.target)** in the **Online Features** tab of the dialog box.

For more information, see Setting Targets for Manipulated Variables.

Run-time weight tuning on manipulated variables

You can now specify weights on the manipulated variables during run time. At the command line, specify the value in the `MVWeights` field of the `mpcmoveopt` object. In the MPC controller blocks, select **Weights on manipulated variables (u,wt)** in the **Online Features** tab of the dialog box.

For more information, see Setting Targets for Manipulated Variables.

Run-time weight tuning and performance monitoring in Multiple MPC Controllers block

You use the Multiple MPC Controllers block to implement gain-scheduled MPC control strategy by switching between multiple MPC controllers. You can now use this block to perform all the tasks that you perform with the MPC Controller block, such as online weight tuning, custom state estimation and performance monitoring.

getEstimator and setEstimator commands to obtain and change state estimation parameters

You can now use `getEstimator` to obtain the Kalman filter gains `L` and `M`, and additional parameters of the following observer equation used by the MPC controller:

```
ym[n|n-1] = Cm*x[n|n-1] + Dvm*v[n]

x[n|n] = x[n|n-1] + M*(ym[n]-ym[n|n-1])

x[n+1|n] = A*x[n|n-1] + Bu*u[n] + Bv*v[n] + L*(ym[n] - ym[n|n-1])
```

Similarly, use `setEstimator` to change the parameters. For more information, see the `getEstimator` and `setEstimator` reference pages.

Compatibility Considerations

`getestim` and `setestim` commands warn and will be removed in a future release. Follow the instructions in the warning message to replace all instances with `getEstimator` and `setEstimator`.

Definition of external MV signal changed

The definition of externally supplied MV signals has been changed from `u[k]` to `u[k-1]`. This implies that MPC controller now expects the external MV signal to be measured at the previous control interval `k-1` and not at the current interval `k`.

Compatibility Considerations

If you enabled the `ext.mv` inport in the MPC Controller or Multiple MPC Controllers block, do the following:

- If the connected signal does not come from the same MPC block, add a unit delay or memory block to the signal so that it is converted from `u[k]` to `u[k-1]`.
- If the connected signal comes directly from the `mv` outport of the same MPC block, you see a warning about algebraic loop. To remove the warning, add a unit delay or memory block in the loop.

There is no incompatibility when you use `mpcmove` at the command prompt.

Unconnected input and output limits inports default changed to match mpc object

You can add inports (`umin`, `umax`, `ymin`, `ymax`) to the MPC controller blocks that you can connect to run-time constraint signals.

- If a channel is unconstrained in the `mpc` object, it remains unconstrained even if the inport is connected and the provided value is ignored.
- If a channel is constrained, the original constraint specified in the `mpc` object is used when the corresponding inport is unconnected.

Compatibility Considerations

Previously, when unconnected, MPC Controller block assumed the online constraint are unbounded (+/- inf). In this release, the simulation output may differ from previous releases because of the change in defaults

R2014a

Version: 4.2

New Features

Bug Fixes

IEC 61131–3 Structured Text generation from MPC Controller and Multiple MPC Controllers blocks using Simulink PLC Coder

The MPC Controller and Multiple MPC Controllers blocks support generation of IEC 61131–3 Structured Text using Simulink PLC Coder. You can verify the generated code using the CoDeSys version 2.3 IDE.

For an example of Structured Text generation for an MPC controller see, Simulation and Structured Text Generation Using PLC Coder.

Reduced RAM usage for C code generated for MPC Controller and Multiple MPC Controllers blocks

The C code generated for the MPC Controller and Multiple MPC Controllers blocks reduces RAM usage. This change includes improved handling of memory allocation. For example, now the generated code does not use dynamic memory allocation, thereby extending support to targets that disallow dynamic memory allocation.

Estimate of data memory size used by deployed MPC controller at run time

You can determine if the data memory size required by an MPC controller exceeds the physical memory of the target system. The report generated by the `review` command now includes a platform-independent estimate of the data memory usage of an MPC controller at run time.

For an example, see Review Model Predictive Controller for Stability and Robustness Issues.

R2013b

Version: 4.1.3

New Features

Bug Fixes

Controller design for plant and disturbance models with internal delays

You can now design model predictive controllers for plant models, input (unmeasured) disturbance models, and output disturbance models that have internal delays. Previously, the software supported only input, output, or transport delays for plant and disturbance models.

When designing the MPC controller, the software discretizes the plant and disturbance models to the controller sample time. The software replaces each model delay of K sampling periods with K poles at $z = 0$. This delay absorption increases the model order, which increases the controller order.

If the models contain significant delays, you must specify an appropriate controller sample time. If the controller sample time is too large, you may not achieve the desired controller performance. However, if you sample a model that contains delays too fast, delay absorption leads to a high-order controller. Such a controller can have a large memory footprint, which can cause difficulty if you generate code for a real-time target. Also, high-order controllers can have numerical precision issues.

For more information regarding internal delays, see Internal Delays. To learn more about specifying a plant model, see Plant Specification. To specify the input disturbance model and the output disturbance model, see `setindist` and `setoutdist`.

Single-precision simulation and code generation using MPC Controller and Multiple MPC Controllers blocks

You can now specify the output data type for the MPC Controller and Multiple MPC Controllers blocks as `single`. This change provides the ability to simulate and generate code for model predictive controllers to be used on single-precision targets. Previously, these blocks supported only double-precision outputs.

To specify the output data type, in the block dialog, use the **Output data type** drop-down list.

For more information, see Simulation and Code Generation Using MPC Controller Block, MPC Controller, and Multiple MPC Controllers.

Conditional execution of MPC Controller and Multiple MPC Controllers blocks using Function-Call Subsystem and Triggered Subsystem blocks

MPC Controller and Multiple MPC Controllers blocks can now inherit the parent subsystem's sample time. Therefore, you can conditionally execute these blocks using the Function-Call Subsystem or Triggered Subsystem blocks.

To specify that the output sample time be inherited, in the block dialog, select the **Block uses inherited sample time (-1)** check box.

Note When you place an MPC controller inside a Function-Call Subsystem or Triggered Subsystem block, you must execute the subsystem at the controller's design sample rate. You may see unexpected results if you use an alternate sample rate.

For more information, see Using MPC Controller Block Inside Function-Call and Triggered Subsystems, MPC Controller, and Multiple MPC Controllers.

R2013a

Version: 4.1.2

Bug Fixes

R2012b

Version: 4.1.1

Bug Fixes

R2012a

Version: 4.1

New Features

Bug Fixes

Compatibility Considerations

Run-Time Preview of Reference and Measured Disturbance Signals with MPC Controller Block

This release introduces the ability to preview signals by using the **ref** and **md** inports of the MPC Controller block and the Multiple MPC Controllers block.

The **ref** inport now accepts an N-by-Ny signal, where N is the number of previewing steps and Ny is the number of plant outputs.

The **md** inport now accepts an N-by-Nmd signal, where N is the number of previewing steps and Nmd is the number of measured disturbances.

You cannot preview if the input signal is a vector, unless Ny or Nmd, as appropriate, is 1.

For more information, see the following examples:

- Improving Control Performance with Look-Ahead (Previewing)
- Chemical Reactor with Multiple Operating Points

Compatibility Considerations

In the current release, if you have models with the MPC Controller block or the Multiple MPC Controllers block, you will see a warning if your blocks contain:

- A **custom reference signal** specified in the MATLAB workspace.
- A **custom disturbance signal** specified in the MATLAB workspace.

Custom Reference Signal Specified in MATLAB Workspace

You must clear this warning. If you ignore the warning, the block will assume that the **ref** signal is zero. This behavior is equivalent to leaving the **ref** inport unconnected.

- **Without Look-Ahead (Previewing) Option.** To eliminate this warning:

   1   Add a From Workspace block to your model.
   2   Specify your reference signal variable name as the **Data** parameter of the From Workspace block.
   3   Connect the output of the From Workspace block to the **ref** inport of the MPC Controller block or the Multiple MPC Controllers block.

- **With Look-Ahead (Previewing) Option.** To eliminate this warning:

   1   Copy the Reference Previewer block from the mpc_preview model and place it in your model. See the Improving Control Performance with Look-Ahead (Previewing) example for more information.
   2   Specify your reference signal variable name as the **Signal** parameter of the Reference Previewer block. Also specify appropriate values for the **Sampling time** and **Number of previewing steps** parameters.
   3   Connect the output of the Reference Previewer block to the **ref** inport of the MPC Controller block or the Multiple MPC Controllers block.

Custom Disturbance Signal Specified in MATLAB Workspace

You must clear this warning. If you ignore the warning, the block will assume that the md signal is zero. This behavior is equivalent to leaving the md inport unconnected.

- **Without Look-Ahead (Previewing) Option.** To eliminate this warning:

  1 Add a From Workspace block to your model.

  2 Specify your disturbance signal variable name as the **Data** parameter of the From Workspace block.

  3 Connect the output of the From Workspace block to the **md** inport of the MPC Controller block or the Multiple MPC Controllers block.

- **With Look-Ahead (Previewing) Option.** To eliminate this warning:

  1 Copy the Measured Disturbance Previewer block from the mpc_preview model, and place it in your model. See the Improving Control Performance with Look-Ahead (Previewing) example for more information.

  2 Specify your measured disturbance signal variable name as the **Signal** parameter of the Reference Previewer block. Also specify appropriate values for the **Sampling time** and **Number of previewing steps** parameters.

  3 Connect the output of the Measured Disturbance Previewer block to the **md** inport of the MPC Controller block or the Multiple MPC Controllers block.

R2011b

Version: 4.0

New Features

Bug Fixes

Compatibility Considerations

C Code Generation Improvements for All Targets with MPC Controller Block

The MPC Controller block has been re-implemented using a MATLAB Function block and now supports code generation for all Simulink Coder targets.

For more information, see the `Code Generation with Simulink Coder` example.

Faster QP Solver Algorithm for Improving MPC Controller Performance

This release implements a new quadratic problem (QP) solver that uses the KWIK algorithm. KWIK is faster and more numerically robust than the previous solver for ill-conditioned QP problems. You can use this solver without default constraints on decision variables.

For more information, see MPC QP Solver.

Run-Time Weight Tuning and Constraint Softening for MPC Controller

This release introduces three new run-time tuning parameters for the MPC Controller block:

- **Weights on plant outputs**
- **Weights on manipulated variables rate**
- **Weight on overall constraints softening**

You can use these parameters to tune the weights on plant outputs, manipulated variables rate, and overall constraint softening. These capabilities are available in real time, without redesigning or re-implementing the MPC controller, and help adjust the controller performance.

For more information, see the `Tuning Controller Weights` example.

You can also use an `mpcmoveopt` object as an input to `mpcmove` to tune the weights and constraints.

For more information, see the following examples:

- `Switching Controllers Based on Optimal Costs`
- `Varying Input and Output Constraints`

Run-Time Monitoring of MPC Controller Performance to Detect When an Optimal Solution Cannot Not Be Found

This release introduces a new outport parameter—**Optimization status** in the MPC Controller block. You can use this outport to monitor the status of the optimization and take the necessary action when an optimal solution cannot be found. For more information, see the `Monitoring Optimization Status to Detect Controller Failures` example.

You can also use the `Info.QPCode` field of the output of `mpcmove` to monitor the status of the optimization.

For more information, see the `mpcmove` reference page.

review Command for Diagnosing Issues with MPC Controller Parameters That Could Lead to Run-Time Failures

You can now use `review` to detect potential stability and robustness issues (both offline and at run time) with an MPC Controller design. The following aspects of the system are inspected:

- Stability of the model predictive controller and the closed loop
- Potential for contradictory settings in the specified constraints and mitigation of an ill-conditioned QP problem by softening constraints
- Validity of QP Hessian matrix

Use this command before implementing the MPC Controller, in conjunction with simulation.

For more information, see the following:

- `review` reference
- `Reviewing Model Predictive Controller Design for Potential Stability and Robustness Issues` example

mpcmove Returns Aligned Time Horizons for Optimal Control, Predicted Output and Estimated State

`mpcmove` now returns `Info` with a time horizon of *t=k,...,k+p*, where *k* is the current time and *p* is the prediction horizon for the following fields:

- `Info.Uopt` — Optimal manipulated variable adjustments
- `Info.Yopt` — Predicted output
- `Info.Xopt` — Predicted state
- `Info.Topt` — Time horizon

You can now plot `Info.Uopt,Info.Yopt` and `Info.Xopt` using `Info.Topt` as the time vector.

For more information, see the `mpcmove` reference page.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
getmpcdata	Still runs	- get - getconstraint - getestim - getindist - getoutdist	Not applicable
pack	Still runs	Not applicable	Not applicable
qpdantz	Warns	quadprog (requires Optimization Toolbox)	Replace all instances of qpdantz with quadprog.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
setmpcdata	Still runs	• set • setconstraint • setestim • setindist • setoutdist	Not applicable

R2011a

Version: 3.3

New Features

Bug Fixes

Support for Custom Constraints on MPC Controller Inputs and Outputs

In addition to upper and lower bounds, you can now specify constraints on linear combinations of an MPC controller inputs ($u(t)$) and outputs ($y(t)$). Specify custom constraints, such as $u1 + u2 < 1$ or $u + y < 2$, in the mpc object using setconstraint.

For more information, see:

- Custom Constraints on Inputs and Outputs
- Custom Constraints in a Blending Process
- MPC Control with Constraints on a Combination of Input and Output Signals example

Ability to Specify Terminal Constraints and Weights on MPC Controller

You can now specify weights and constraints on the terminal predicted states of an MPC controller.

Using terminal weights, you can achieve infinite horizon control. For example, you can design an unconstrained MPC controller that behaves in exactly the same way as a Linear-Quadratic Regulator (LQR). You can use terminal constraints as an alternative way to achieve closed-loop stability by defining a terminal region.

You can specify both weights and constraints using the setterminal command.

For more information, see:

- Terminal Weights and Constraints
- Using Terminal Penalty to Provide LQR Performance
- Implementing Infinite-Horizon LQR by Setting Terminal Weights in a Finite-Horizon MPC Formulation example

Ability to Access Optimal Cost and Optimal Control Sequence

This release introduces two new parameters **Enable optimal cost outport** and **Enable control sequence outport** in the MPC Controller block. Using these parameters, you can access the optimal cost and control sequence along the prediction horizon. This information helps you analyze control performance.

You can also access the optimal cost and control sequence programmatically using the new Cost and Yopt fields, respectively, of the structure info returned by mpcmove.

For more information on using optimal cost and control sequence, see the following examples:

- MPC Control with Input Quantization Based on Comparing the Optimal Costs
- Analysis of Control Sequences Optimized by MPC on a Double Integrator System

R2010b

Version: 3.2.1

No New Features or Changes

R2010a

Version: 3.2

New Features

Bug Fixes

New Ability to Analyze SISO Generalized Predictive Controllers (GPC)

You can now use `gpc2mpc` to convert your SISO GPC controller to an MPC controller. Analyze and simulate the resulting MPC controller using available Model Predictive Control Toolbox commands.

For more information, see the `gpc2mpc` reference page.

R2009b

Version: 3.1.1

Bug Fixes

R2009a

Version: 3.1

New Features

Bug Fixes

New Sensitivity Analysis to Determine Effect of Weights on Tuning MPC Controllers

You can now perform sensitivity analysis to determine the effect of weights on the closed-loop performance of your system. You can perform sensitivity analysis using the following:

- MPC Tuning Advisor. See Tuning Advisor in the *Model Predictive Control User's Guide*.
- `sensitivity` command. See the sensitivity reference page.

R2008b

Version: 3.0

New Features

Bug Fixes

New Multiple MPC Controllers Block in the Model Predictive Control Toolbox Simulink Library

You can now use the Multiple MPC Controllers block in Simulink software to control a nonlinear process over a range of operating points. You include an MPC controller for each operating point in the Multiple MPC Controllers block and specify switching between these controllers in real-time based on the input scheduling signal to the block. If you need to change the design of a specific controller, you can open the MPC Design Tool GUI directly from the Multiple MPC Controllersblock.

During model simulation, Model Predictive Control Toolbox provides bumpless transfer when the system transitions between operating points.

To learn more about configuring the new block, see the Multiple MPC Controllers block reference page.

Tested Code Generation Support for Real-Time Workshop Target Systems

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

- Generic Real-Time Target
- Real-Time Workshop Embedded Coder™
- Real-Time Windows Target
- Rapid Simulation Target
- Target Support Package FM5
- xPC Target (known as Simulink Real-Time™ as of R2014a)
- dSpace Target
- Target for Infineon TriCore

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

- Embedded Target for TI C2000 DSP
- Embedded Target for TI C6000 DSP
- Target Support Package IC1 (for Infineon C166)
- Tornado (VxWorks) Real-Time Target

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

Ability to Design Controllers with Time-Varying Weights and Constraints Using the GUI

While you design an MPC controller using the MPC Design Tool graphical user interface (GUI), you can specify time-varying weights and constraints for manipulated variables, rate of change of

manipulated variables, and output variables. In the previous version, you could only specify the time-varying weights and constraints at the command line.

Furthermore, you can load an MPC controller with time-varying information from the command line into the MPC Design Tool GUI.

To learn more about the new options in the MPC Design Tool GUI, see the Model Predictive Control Toolbox documentation.

R2008a

Version: 2.3.1

No New Features or Changes

R2007b

Version: 2.3

New Features

New Option for Specifying Time-Varying Constraints

You can now configure the Model Predictive Controller block in Simulink to accept time-varying constraint signals that are generated by other blocks. To add inports to which you can connect time-varying constraint specifications, select the new **Enable input port for input and output limits** check box in the MPC Controller block. See also the `mpcvarbounds` demo.

In the previous version, you could only specify the constraints during the design phase and these constraints remained constant for the duration of the simulation.

For more information about the new **Enable input port for input and output limits** check box in the Model Predictive Controller block, see the MPC Controller block reference page.

Ability to Specify Nondiagonal Q and R Weight Matrices in the Cost Function

You can now specify off-diagonal weights in the cost function. In the previous release, only diagonal Q and R matrices were supported.

To learn more about specifying off-diagonal weights, see the discussion about weights in the MPC Controller block reference pages.

To access a new demo that shows how to use nondiagonal weight matrices, type the following command at the MATLAB prompt:

```
showdemo('mpcweightsdemo')
```

R2007a

Version: 2.2.4

Bug Fixes

R2006b

Version: 2.2.3

No New Features or Changes

R2006a

Version: 2.2.2

New Features

Bumpless Transfer Added to MPC Block

Bumpless transfer between manual and automatic operation or from one controller to another has been added to the Model Predictive Controller block in Simulink. This block now allows feedback of the true manipulated variable signals, which allows the controller to maintain an accurate state estimate during periods when its calculated adjustments are not being sent to the plant. For example, the controller's output might be ignored during a startup period or during temporary intervention by a (simulated) plant operator. If the controller assumes that its adjustments are being implemented (the default behavior), its state estimate will be incorrect, leading to a "bump" when the controller is reconnected to the plant. A tutorial example has been added to the documentation.

New Bumpless Transfer Demo

A new demo illustrating bumpless transfer has been added to the toolbox.

R14SP3

Version: 2.2.1

No New Features or Changes

R14SP2

Version: 2.2

No New Features or Changes