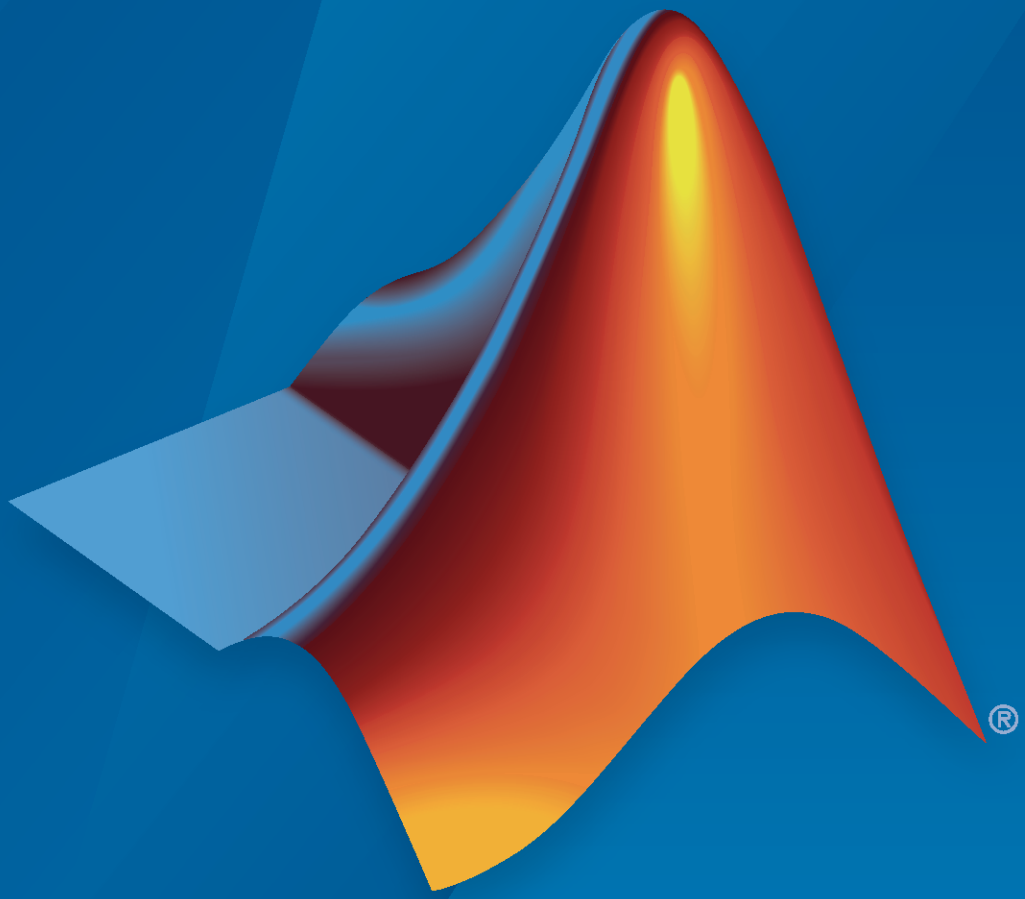
# Optimization Toolbox™ Release Notes



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

*Optimization Toolbox™ Release Notes*

# Contents

## R2023a



## R2022b



## R2022a

## R2021b



## R2021a



## R2020b

## R2020a



## R2019b

## R2019a



## R2018b

## R2018a



## R2017b

## R2017a



## R2016b

## R2016a



## R2015b



## R2015a

## R2014b



## R2014a

## R2013b



## R2013a



## R2012b



## R2012a



## R2011b

## R2011a



## R2010b



## R2010a

## R2009b



## R2009a



## R2008b

## R2008a



## R2007b



## R2007a



## R2006b

## R2006a

**Bug Fixes**

## R14SP3

# R2023a

## Linear and Nonlinear Constraints in Nonlinear Least Squares: Solve Least Squares Problems with Constraints

The `lsqnonlin` and `lsqcurvefit` solvers now accept linear and nonlinear constraints. You can use these constraints by calling the solvers directly, by solving a least-squares problem using the problem-based approach, or by using the **Optimize** Live Editor task. For examples, see the function reference pages.

The solvers can solve constrained problems using the new `'interior-point'` algorithm. Internally, this algorithm reformulates a least-squares problem for the `fmincon` interior-point solver. Usually, this reformulation leads to a solution more efficiently than specifying a problem for `fmincon` directly. See “Compare lsqnonlin and fmincon for Constrained Nonlinear Least Squares”.

If you specify linear or nonlinear constraints and do not specify an algorithm, the solvers switch to the `'interior-point'` algorithm. If you specify these constraints and specify another algorithm, the solvers throw an error.

## Compatibility Considerations

- For solver-based optimization, the `'interior-point'` algorithm does not accept functions as names, but instead requires that the objective and nonlinear constraint functions are function handles. To update existing code to use the `'interior-point'` algorithm, specify the objective function using a function handle such as `@objfun` instead of a function name such as `'objfun'`.
- The default solver for problem-based least-squares with linear or nonlinear constraints has changed from `fmincon` to `lsqnonlin`. To have `solve` or `prob2struct` use `fmincon` as the solver, specify the `Solver` name-value argument, such as

  ```
  [sol,fval] = solve(prob,x0,Solver="fmincon")
  ```

## Clarified plots and iterative display in problem-based maximization

The `solve` function returns more informative plots and iterative display for maximization problems. Previously, many plots and iterative display showed the negative of the objective function, because, internally, solvers maximize by minimizing the negative of the objective function. Also, linear and quadratic programming problems did not include any additive constants in the iterative display or plots. Now the plots and iterative display return the expected values for both maximization and minimization problems.

## Compatibility Considerations

For problem-based maximization:

- Iterative display shows the true (not negated) objective function value.
- Custom plot functions and output functions now have the true (not negated) objective function value in the `optimValues.fval` field.
- For solvers that use `optimValues.gradient`, the gradient has its sign flipped from the previous value.
- `optimValues.searchdirection` has a flipped sign from the previous value.

Some functions have minor updates in their iterative display.

- The `fsolve` function has the header `||f(x)||^2` instead of `f(x)`.
- The `fminsearch` function has the header `f(x)` instead of `min f(x)`.
- `lsqlin` has the same headers as `quadprog` except that it uses `Resnorm` instead of `f(x)`.

## fminbnd and fminsearch solvers supported in problem-based optimization

The `solve` and `prob2struct` functions accept `"fminbnd"` and `"fminsearch"` as values in the `Solver` name-value argument. The `fminbnd` solver can solve nonlinear optimization problems that have one bounded scalar variable. The `fminsearch` solver can solve unconstrained multidimensional nonlinear optimization problems. The `solvers` function, which lists the available solvers for a problem, includes these solvers for appropriate problems.

## Improvements in intlinprog and linprog

Internal scaling factors in `intlinprog` and in the `linprog 'dual-simplex'` and `'interior-point'` algorithms are now restricted to be powers of 2. This change avoids floating-point rounding errors. Along with other code changes, the solvers can exhibit improved reliability and performance.

## Compatibility Considerations

These changes can cause the solution path taken by these algorithms to differ from that of previous versions. The changes do not always lead to better performance. There is no provision for obtaining the previous behavior.

## Optimal Trajectory Example: More Efficient Problem Formulation, Extension to Variable Mass

The example “Discretized Optimal Trajectory, Problem-Based” has the following improvements:

- Fewer problem variables. The variables representing position and velocity at each time are now inferred by integrating the equations of motion, rather than being problem variables.
- Effect of expelled mass. The object's mass decreases as it expels fuel. The example shows how to account for this effect in the trajectory, and how to choose a good starting point for the solver.

# R2022b

## Enhanced Analysis in fcn2optimexpr: Speed problem creation and solution

The `fcn2optimexpr` function has enhanced analysis of problems. In particular, `for` loops are faster than before, and nested `for` loops are much faster. In order to realize this speed gain, you must convert your `for` loops to functions, as described in Static Analysis of Optimization Expressions.

For more information about enhanced analysis, which is named static analysis, see Static Analysis of Optimization Expressions. For examples of converting `for` loops to functions, see Create `for` Loop for Static Analysis and Convert Constraints in `for` Loops for Static Analysis.

## solvers Function: Find the default and available solvers for a problem

For a problem expressed as an `OptimizationProblem` or `EquationProblem`, the `solvers` function lists the default solver and all valid solvers for the problem. For example,

```
[defaultsolver,validsolvers] = solvers(prob)
```

You can call any valid solver by using the `Solver` name-value argument in `solve` or `prob2struct`. For example, if `fmincon` is a valid solver for `prob`:

```
[sol,fval] = solve(prob,x0,Solver="fmincon")
% or
problem = prob2struct(prob,x0,Solver="fmincon")
```

If you have Global Optimization Toolbox, the lists of default and valid solvers include Global Optimization Toolbox solvers.

## optim.coder.infbound: Express infinite bounds for code generation

For code generation where the target does not accept `Inf` as a value of a bound, use `optim.coder.infbound` instead. For example,

```
lb = [-optim.coder.infbound,0]; % lb(1) = -Inf, lb(2) = 0
ub = optim.coder.infbound(1,2); % ub has two elements, all Inf
```

For details, see the function reference page.

# R2022a

## Problem-Based Optimize Live Editor Task: Solve optimization problems or systems of equations using a visual interface

The **Optimize** Live Editor task now supports the Problem-Based Optimization Workflow. **Optimize** also supports the problem-based approach for Global Optimization Toolbox single-objective and multiobjective solvers. Using problem-based **Optimize** you can:

- Create optimization variables, expressions, and problems.
- Create linear or nonlinear objectives, constraints, or equations using built-in function templates.
- Optionally, specify the solver and options.
- Run the solver directly from the task.
- Export the resulting MATLAB® code.

To launch the task, create a Live Editor window, then select **Task > Optimize** or **Insert > Task > Optimize**. Then choose the problem-based approach.

For an example, see Get Started with Problem-Based Optimize Live Editor Task. For tips, see Use Problem-Based Optimize Live Editor Task Effectively.

## LBFGS Algorithm in fminunc: Optimize large problems

To save memory and gain speed in large `fminunc` problems, use the `"quasi-newton"` algorithm and set the `HessianApproximation` option to `"lbfgs"`. For an example of the speed gains, see Solve Nonlinear Problem with Many Variables. For algorithm details, see `fminunc` `quasi-newton` Algorithm.

## Compatibility Considerations

The option name has been renamed from *HessUpdate* to `HessianApproximation`. For details, see `fminunc` `options`.

## fcn2optimexpr Arguments: Affect conversion and obtain details of conversion

The `fcn2optimexpr` function gains two name-value arguments that can affect the conversion process and give details of the conversion:

- `Analysis` — When `"on"` (default), `fcn2optimexpr` analyzes the function being converted to see whether it is composed entirely of supported operations (see Supported Operations for Optimization Variables and Expressions). If so, `fcn2optimexpr` uses the operations to create the optimization expression, and the resulting expression supports automatic differentiation (see Automatic Differentiation Background). Furthermore, `solve` uses the most appropriate solver by default, as described in `Solver`. If the function is not composed entirely of supported operators, the resulting expression is a black box. In this case, the expression does not support automatic differentiation, and `solve` has restricted solver choice, so can fail to use the most efficient solver.
- `Display` — When `"on"`, reports the results of `Analysis`.

## Compatibility Considerations

When a function is composed entirely of supported operations, by default `fcn2optimexpr` returns a different expression than previously. The returned expression supports automatic differentiation and all applicable solvers.

The function analysis can take some time. This time can be noticeable for complicated functions with nested loops.

To obtain the previous behavior, set `Analysis="off"`.

## Functionality being removed or changed

**Relative tolerance change for intlinprog**
*Behavior change*

`intlinprog` changes the stopping condition associated with the `RelativeGapTolerance` option. Now `intlinprog` stops if the relative difference between the internally calculated upper (`U`) and lower (`L`) bounds on the objective function is less than or equal to `RelativeGapTolerance`:

`(U – L)/(|U| + 1) <= RelativeGapTolerance`.

Previously, when `L` had a large magnitude, `intlinprog` automatically modified the tolerance:

tolerance = `min(1/(1 + |L|), RelativeGapTolerance)`.

This change makes the stopping condition easier to understand and apply.

To obtain approximately the previous behavior when the objective function value has large magnitude, set the option to the value `min(1/(1 + |L|), RelativeGapTolerance)`, where `|L|` is the magnitude of the objective function for the same problem without integer constraints (`intcon = []`).

# R2021b

## fmincon 'interior-point' Algorithm: Obtain feasible solutions using a new feasibility mode

The `fmincon` `'interior-point'` algorithm has an updated feasibility routine. For problems where `fmincon` has difficulty reaching a feasible solution, this routine sometimes enables `fmincon` to succeed. The routine usually works better when you also set the `SubproblemAlgorithm` option to `'cg'`. To use the routine, set the new `EnableFeasibilityMode` option to `true` using `optimoptions`.

```
options = optimoptions('fmincon',...
    'Algorithm','interior-point',...
    'EnableFeasibilityMode',true,...
    'SubproblemAlgorithm','cg');
[x,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

For details, see Feasibility Mode. For an example, see Obtain Solution Using Feasibility Mode.

## Functionality being removed or changed

**Problem-based behavior change for norm(sum of squares)^2**
*Behavior change*

When an optimization expression has the form `norm(expression)^2`, internal simplification routines now convert the expression to a sum of squares. Therefore, `solve` and `prob2struct` can now use the `lsqlin` and `lsqnonlin` solvers for this type of objective function when the problem constraints are compatible (linear constraints for `lsqlin` or bound constraints for `lsqnonlin`). See Write Objective Function for Problem-Based Least Squares.

**Problem-based behavior change for prob2struct with integer constraints**
*Behavior change*

When a problem has integer constraints and a nonlinear objective function, the default behavior of the `prob2struct` function now depends on whether you have Global Optimization Toolbox installed.

- If Global Optimization Toolbox is installed, `prob2struct` returns a `ga` problem.
- If Global Optimization Toolbox is not installed, `prob2struct` throws an error.

Previously, when a problem included integer constraints and a nonlinear objective function, the result was a `coneprog`, `fmincon`, `fminunc`, `lsqlin`, `lsqnonlin`, or `quadprog` problem structure. However, these solvers cannot solve problems with integer constraints.

To obtain the previous behavior, set the `Solver` name-value argument to the appropriate solver name. When you do, `prob2struct` warns that the resulting problem cannot be solved with integer constraints. For example, for a quadratic programming problem, enter

```
problem = prob2struct(prob,"Solver","quadprog");
```

For details, see Integer Constraints in Nonlinear Problem-Based Optimization.

# R2021a

## Automatic Differentiation: Solve nonlinear least-squares and equation problems using automatically computed derivatives in the problem-based approach

Automatic Differentiation (AD) now applies to problem-based nonlinear least-squares and systems of nonlinear equations, as well as to general nonlinear optimization problems. For added speed with these problem types, solvers can now use forward AD as well as reverse AD. New option values in `solve` and `prob2struct` allow you to choose the AD type. For the default AD type for various problems and solvers, see the `'ObjectiveDerivative'` argument.

Problem-based expressions can use more trigonometric and hyperbolic functions, including `sec`, `asec`, `sech`, and `asech`. For details, see Automatic Differentiation Background and Supported Operations on Optimization Variables and Expressions.

## Warm Start quadprog and lsqlin: Solve problems faster by reusing data structures

The `quadprog 'active-set'` algorithm and `lsqlin 'active-set'` algorithm can update and use warm start data structures. These structures speed solving problems that are similar to previously solved problems. Warm start is available for `quadprog` and `lsqlin` code generation as well. Create a warm start object using the `optimwarmstart` function. For details, see Warm Start Best Practices, Warm Start `quadprog`, and the function reference pages.

## Code Generation for Linear Least Squares: Generate C code for linear least squares problems

The `lsqlin 'active-set'` algorithm now supports code generation. For examples and details, see Code Generation in Linear Least Squares: Background and Generate Code for `lsqlin`.

## Interior-Point fmincon Algorithm: Solve constrained nonlinear problems faster

The `fmincon 'interior-point'` algorithm has an added barrier parameter update algorithm. Set the barrier parameter algorithm using the new `BarrierParamUpdate` option. In tests, the solver is often faster for problems with inequality constraints (including bound constraints) when this option is set to `'predictor-corrector'`. To obtain the previous `fmincon` behavior, set the `BarrierParamUpdate` option to its default value, `'monotone'`.

For details, see fmincon Interior Point Algorithm.

## Second-Order Cone Programming: Create and solve problems with second-order cone constraints in the problem-based approach

Problem-based optimization now uses `coneprog` internally to solve second-order cone programming problems when you call `solve` or `prob2struct`. Create a problem with a linear objective function and optionally with bounds or linear constraints. Specify a second-order cone constraint using the newly-supported `norm` function using the syntax

```
norm(linear_expression1) + constant <= linear_expression2;
```

Here, `linear_expression` represents a linear expression in optimization variables. You can also specify a cone constraint as the square root of a sum of squares of optimization variables. See Write Constraints for Problem-Based Cone Programming. For an example, see Minimize Energy of Piecewise Linear Mass-Spring System Using Cone Programming, Problem-Based.

## New Algorithms in Second-Order Cone Programming: Solve problems more quickly and with greater numerical stability

The `coneprog` function has a new option named `LinearSolver` that can help you to achieve greater speed and stability in problems with large cones or large, sparse constraint matrices with some dense columns. By default, `coneprog` chooses an internal solution algorithm based on your problem. For more information, see Compare Speeds of `coneprog` Algorithms, the `coneprog` options section, and Second-Order Cone Programming Algorithm.

## Compatibility Considerations

To obtain the same behavior as the previous release, set the `LinearSolver` option to `'augmented'`.

## Functionality Being Removed or Changed

### Optimization App Removed

*Errors*

The Optimization app (`optimtool`) has been removed. For a visual interface to solvers, use the **Optimize** Live Editor task.

### prob2struct Options Name-Value Pair Removed

*Errors*

The `Options` name-value pair has been removed from `prob2struct`. To modify options, edit the resulting `problem` structure. For example,

```
problem.options = optimoptions('fmincon',...
    'Display','iter','MaxFunctionEvaluations',5e4);
% Or, to set just one option:
problem.options.MaxFunctionEvaluations = 5e4;
```

The `Options` name-value pair was removed because it can cause ambiguity in the presence of automatic differentiation.

### mapSolution Removed

*Errors*

The `mapSolution` function has been removed. To obtain the indices of optimization variables when you convert a problem using `prob2struct`, use `varindex`. For an example showing how to create a solution structure from a converted problem, see Solve Problem Using Both Approaches.

### Two coneprog lambda Structures Renamed

*Behavior change*

The `coneprog` `lambda` output argument fields `lambda.eq` and `lambda.ineq` have been renamed to `lambda.eqlin` and `lambda.ineqlin`, respectively. This change causes the `coneprog` `lambda` structure fields to have the same names as the corresponding fields in other solvers.

Also, when `coneprog` returns exit flag `–2`, `–3`, or `–10`, the `lambda` output argument is now empty. Previously in these cases, the `lambda` output argument was a structure with empty fields.

**Optimize Bound Behavior Change**
*Behavior change*

The `Optimize` Live Editor task no longer performs scalar expansion on a bound specified as `From workspace` and as a scalar. Previously, if your workspace variable was a scalar and the problem had multiple variables, **Optimize** applied the scalar bound to all problem variables. Now, **Optimize** treats a scalar value `From workspace` as applying to the first variable alone, with no scalar expansion.

You can have a single scalar value apply to all variables by choosing `All bounds the same` instead of `From workspace`.

# R2020b

## Second-Order Cone Programming: Solve problems with second-order cone constraints

The `coneprog` function solves problems with a linear objective function, second-order cone constraints, bounds, and linear constraints. Several problem types can be reformulated into cone programming problems; see Convert Quadratic Constraints to Second-Order Cone Constraints and Convert Quadratic Programming Problem to Second-Order Cone Program. Create second-order cone constraints using the `secondordercone` function. For details, see the function reference pages and Quadratic Programming and Cone Programming category.

## Automatic Differentiation in Problem-Based Optimization: Use gradients of optimization expressions and constraints automatically

Problem-based optimization can automatically calculate and use gradients of objective and constraint expressions. These automatic gradients can provide faster, more reliable calculations than the previous default, finite-difference approximations.

- Optimize constrained or unconstrained nonlinear objective expressions using automatic differentiation by calling `solve`.
- Generate first derivative code automatically for a nonlinear objective and constraints by calling `prob2struct`.

For automatic differentiation details, see Automatic Differentiation Background. For an example, see Effect of Automatic Differentiation in Problem-Based Optimization.

## Optimize Live Editor Task: Create and run optimization problems using a visual interface

The `Optimize` Live Editor task helps you formulate and run optimization and equation-solving problems using a visual interface. **Optimize** applies to all MATLAB optimization solvers, all Optimization Toolbox solvers, and, with a license, all Global Optimization Toolbox solvers, except for `fseminf`, `GlobalSearch`, and `MultiStart`. For an example, see Optimize Live Editor Task with `fmincon` Solver. For details, see the `Optimize` reference page.

## Compatibility Considerations

The **Optimize** Live Editor task replaces the Optimization app (`optimtool`) as a visual interface for creating and solving optimization problems. As noted since R2015b, the Optimization app will be removed in a future release.

**Optimize** provides only two outputs from the solver, typically called `x` and `fval` in the documentation. Specifically, **Optimize** does not provide the `exitflag` and `output` structure information. To obtain more information from the solver, use the generated code to rerun the solver while requesting more outputs.

## Code Generation for Nonlinear Least Squares and Equation Solving: Generate C code for nonlinear least squares and systems of nonlinear equations

The `'levenberg-marquardt'` algorithm of the `lsqcurvefit`, `lsqnonlin`, and `fsolve` solvers now supports code generation. For examples and details, see Code Generation in Nonlinear Least

Squares: Background, Generate Code for `lsqcurvefit` or `lsqnonlin`, Code Generation in Nonlinear Equation Solving: Background, and Generate Code for `fsolve`.

## Nonlinear Least Squares: 'levenberg-marquardt' algorithm accepts bounds

The `'levenberg-marquardt'` algorithm in the `lsqcurvefit` and `lsqnonlin` solvers now accepts bounds. For details, see the function reference pages and Bound Constraints in Levenberg-Marquardt Method.

## fmincon bestfeasible Field in output Structure: Obtain the best feasible point evaluated in solution process

The `fmincon` solver can return a point that is not the best feasible point that it evaluates during its solution process. Instead, it can return a point with better first-order optimality, or even an infeasible point. The `output` structure now has a `bestfeasible` field that contains a structure with the best feasible point evaluated, if any. For an example, see Obtain Best Feasible Point.

C code generated from `fmincon` does not contain the `bestfeasible` field in a returned `output` structure.

## Names in MPS Files: mpsread optionally returns variable and constraint names for continuous and mixed-integer linear programming problems

You can obtain the names of variables and constraints in the `problem` structure when converting an MPS file using `mpsread`. These names can help you match the MPS inputs to your solver outputs. To obtain the names, call `mpsread` with `'ReturnNames'` set to `true`. For an example, see Obtain Variable and Constraint Names.

## Heuristics for intlinprog: Obtain feasible points faster

`intlinprog` has a new heuristic for improving integer-feasible points called 2-opt. This heuristic runs when the `Heuristics` option is `'basic'`, `'intermediate'`, or `'advanced'`. For details, see Heuristics for Finding Feasible Solutions.

## Compatibility Considerations

Because `intlinprog` can find feasible solutions faster, the solution process can take different iterations than before. This can affect the time and number of iterations for finding the solution, either an increase or a decrease.

## Functionality Being Removed or Changed

### prob2struct Options Name-Value Pair Will Be Removed

*Still runs*

The `Options` name-value pair will be removed from `prob2struct` in a future release. To modify options, edit the resulting `problem` structure. For example,

```
problem.options = optimoptions('fmincon',...
    'Display','iter','MaxFunctionEvaluations',5e4);
% Or, to set just one option:
problem.options.MaxFunctionEvaluations = 5e4;
```

The `Options` name-value pair will be removed because it can cause ambiguity in the presence of automatic differentiation.

**prob2struct Creates Default Options**
*Behavior change*

When `prob2struct` creates a `problem` structure, `problem.options` now contains the default options for the solver. Previously, `problem.options` was empty, `[]`. The new behavior enables you to edit just one option easily using tab-completion, such as

```
problem.options.StepTolerance = 1e-9;
```

**Generated Code Uses Column Major Order for Matrices**
*Behavior change*

Code generation now forces matrices and arrays to have column major ordering. For most problems, this ordering significantly improves performance.

# R2020a

## Problem-Based Optimization Functions: Use elementary functions for objective functions or constraints

Problem-based optimization now supports elementary functions and their inverses, including:

- `sqrt`
- `exp`
- `log`
- `sin`
- `cos`
- `tan`
- `asin`
- `acos`
- `atan`
- `sinh`
- `cosh`
- `tanh`
- `asinh`
- `acosh`
- `atanh`

Problem-based optimization also supports `diff`, `cumsum`, and `cumprod`. For a complete list of supported operations, see Supported Operations on Optimization Variables and Expressions. For an example using the `exp` function as well as polynomials, see Solve Constrained Nonlinear Optimization, Problem-Based.

## Quadratic Programming and Linear Least Squares: Active-set algorithm solves dense problems

The `'active-set'` algorithm of the `quadprog` and `lsqlin` solvers gives fast, accurate solutions to medium-sized problems. The algorithm uses dense linear algebra, not sparse, so is not suitable for very large problems. For an example, see Quadratic Programming with Many Linear Constraints.

## Code Generation for Quadratic Problems: Generate C code for problems with linear constraints and quadratic objectives

The `'active-set'` algorithm of the `quadprog` solver supports code generation. For examples and details, see Code Generation for `quadprog`.

## Heuristics for intlinprog: Obtain feasible points faster

`intlinprog` has new heuristics for finding and improving integer-feasible points.

- trivial heuristic — Runs unless the `Heuristics` option is `'none'` or you provide an initial integer-feasible point.

- 1-opt — Runs when the `Heuristics` option is `'basic'`, `'intermediate'`, or `'advanced'`.
- ZI round — Runs when the `Heuristics` option is `'basic'`, `'intermediate'`, or `'advanced'`.

For details, see Heuristics for Finding Feasible Solutions.

## Compatibility Considerations

Because `intlinprog` can find feasible solutions faster, the solution process can take different iterations than before. This can affect the time and number of iterations for finding the solution, either an increase or a decrease.

## Infeasibility Analysis Example: Identify conflicting linear constraints by finding irreducible infeasible or maximal feasible subsets

The example Investigate Linear Infeasibilities shows how to examine which linear constraints cause a problem to be infeasible.

# R2019b

## Problem-Based Equation Solving: Solve systems of equations using optimization variables

You can now solve systems of equations, both linear and nonlinear, with or without bounds, in the problem-based framework. Create optimization variables, create equalities from these variables, and include the equalities in an equation problem you create by calling `eqnproblem`. For a basic example, see Solve Nonlinear System of Equations, Problem-Based. For details, see Systems of Nonlinear Equations.

## Problem-Based Nonlinear Least-Squares: Solve nonlinear least-squares problems using optimization variables

You can now solve nonlinear least-squares problems, with or without bounds, in the problem-based framework. For a basic example, see Nonlinear Least-Squares, Problem-Based. For details, see Nonlinear Least Squares (Curve Fitting).

## fmincon Code Generation: Generate C code for nonlinear constrained optimization (requires MATLAB Coder)

To obtain C code for solving a constrained nonlinear optimization problem, call `codegen` for the `fmincon 'sqp'` algorithm. For examples and details, see Code Generation in `fmincon`.

## Functionality Being Removed or Changed

**OptimizationConstraint split into OptimizationEquality and OptimizationInequality**
*Behavior change*

When you use a comparison operator <=, >=, or == on an optimization expression, the result is no longer an `OptimizationConstraint` object. Instead, the equality comparison == returns an `OptimizationEquality` object, and an inequality comparison <= or >= returns an `OptimizationInequality` object. You can use these new objects for defining constraints in an `OptimizationProblem` object, exactly as you would previously for `OptimizationConstraint` objects. Furthermore, you can use `OptimizationEquality` objects or equalities in `OptimizationConstraint` objects to define equations for an `EquationProblem` object.

**optimconstr split into optimeq and optimineq**
*Behavior change*

You can now specify an empty optimization constraint expression by using `optimeq` or `optimineq`, as well as by using `optimconstr`. The functions work the same, except that `optimeq` requires the constraint expressions to be equalities, and `optimineq` requires the constraint expressions to be inequalities. `optimeq` returns an `OptimizationEquality` object, `optimineq` returns an `OptimizationInequality` object, and `optimconstr` returns an `OptimizationConstraint` object. All of these objects are suitable for use as constraints in optimization problem objects, and equalities are suitable for use as equations in equation problem objects. Use whichever functions you prefer.

**showconstr, showexpr, showproblem, showvar, writeconstr, writeexpr, writeproblem, and writevar are not recommended**
*Still runs*

The `showconstr`, `showexpr`, `showproblem`. `showvar`, `writeconstr`, `writeexpr`, `writeproblem`, and `writevar` functions are not recommended. Instead, use the `show` and `write` functions. By using `show` and `write`, you obtain all of the functionality of the named functions, and do not have to remember the various function names. The display functions `showbounds` and `writebounds` remain recommended, because `show` and `write` do not duplicate their functionality.

There are no plans to remove the named functions at this time.

**Simplified Parallel Deployment**
*Behavior change*

If you deploy code that calls an optimization solver, and want the solver to use parallel computing, you no longer need to create a parallel pool explicitly in your code.

# R2019a

## Problem-Based Nonlinear Optimization: Express nonlinear optimization problems using optimization variables

You can now formulate and solve nonlinear problems using the problem-based approach. To set up a problem using this approach, see Problem-Based Optimization Setup. For examples, see Problem-Based Nonlinear Optimization. To convert nonlinear functions to optimization expressions, use `fcn2optimexpr`.

## Problem-Based Expressions: Rational expression support

You can now use any rational expression in optimization variables for objective or constraint expressions. A rational expression is a quotient of polynomials, including noninteger coefficients. For any other type of expression, convert a program or anonymous function using `fcn2optimexpr`. For examples, see Problem-Based Nonlinear Optimization.

Additionally, the `mean` function now works with optimization variables, and produces an optimization expression. Both `mean(x)` and `mean(x,dim)` are valid syntaxes for an optimization variable `x`.

## Problem-Based Conversion to Solver-Based: Use the varindex function and new parameters for prob2struct

To support the conversion of problems from a problem-based formulation to solver-based, the `varindex` function gives the mapping between optimization variables and indices of the associated solver-based variable. For examples, see Include Derivatives in Problem-Based Workflow and Output Function for Problem-Based Optimization.

To support the conversion of nonlinear problems, `prob2struct` accepts an initial point argument `x0`. Also, `prob2struct` has new name-value pair arguments related to file names and the folder for exported objectives, nonlinear constraints, and supporting functions. For details, see the `prob2struct` function reference page.

## Mixed-Integer Linear Programming Branching: Improved performance with the new default branching method

Reliability branching, an internal algorithm in `intlinprog`, has improved performance. Similarly, the performance of the strong pseudocost branching algorithm (`'strongpscost'`) is improved. As a result, the default value of the `'BranchRule'` option is now `'reliability'`. The `solve` function for the problem-based approach inherits this behavior for mixed-integer linear programming problems.

## Compatibility Considerations

Reliability branching typically performs better than the previous default algorithm, maximum pseudocost (`'maxpscost'`). However, `'maxpscost'` might work better for some problems. To obtain the previous behavior, set the `'BranchRule'` option to `'maxpscost'` using `optimoptions`.

## Mixed-Integer Linear Programming Heuristics: Solve problems faster using updated heuristics

`intlinprog` has improved heuristics implementations, which enable the solver to find the first feasible and optimal solutions faster. Also, heuristics run during branch-and-bound as well as at the root node. For details, see Heuristics for Finding Feasible Solutions.

Iterative display now shows an "h" at the end of each line where heuristics lead to a new integer-feasible solution. Also, every newly found integer-feasible solution leads `intlinprog` to call any output functions and plot functions that you specify.

The `solve` function for the problem-based approach inherits this behavior for mixed-integer linear programming problems.

## Compatibility Considerations

Although the new heuristics implementations generally improve solver performance, occasionally the solution time increases.

## Functionality Being Removed or Changed

**OptimizationExpression gains Variables property**
*Behavior change*

`OptimizationExpression` objects now have a read-only `Variables` property. This property relates only to variables in the expression, not to all the variables in the problem.

**New exit flags for the linprog 'dual-simplex' algorithm and intlinprog**
*Behavior change*

The `linprog 'dual-simplex'` algorithm and `intlinprog` can now return exit flags 3 and –9. `solve` can also return these exit flags for linear programs or mixed-integer linear programs. Previously, these solvers would throw an error for the corresponding exit conditions. For details, see the solver function reference pages.

# R2018b

## Optimization Modeling: Use variable expressions to represent quadratic objectives

The problem-based approach now applies to problems with quadratic objective functions and linear constraints. Previously, a problem-based objective function could be only linear. See Problem-Based Optimization Setup.

For examples using the problem-based approach with quadratic objectives, see Quadratic Programming and Linear Least Squares.

## Optimization Solving: Solve problems with quadratic objectives and linear constraints using an automatically selected solver

`solve` now applies to problems with quadratic objectives and linear constraints. For these problems, `solve` internally calls `quadprog`, or, for linear least-squares problems, `lsqlin` or, if specified in the `solver` option, `lsqnonneg`.

For examples solving quadratic problems using the problem-based approach, see Quadratic Programming and Linear Least Squares.

## intlinprog Performance: Solve problems faster with enhanced preprocessing and branching

The `intlinprog` solver has improved performance due to algorithm changes. These changes include:

- Stronger clique cuts during cut generation (the `CutGeneration` option).
- Inclusion of objective GCD reasoning during branch-and-bound.
- More frequent use of the simple rounding heuristic during branch-and-bound.
- More efficient implementation of `BranchRule` options `'strongpscost'` and `'reliability'`.

## quadprog and lsqlin Options: Choose dense or sparse linear solver explicitly

The `'LinearSolver'` option specifies the underlying linear algebra type for the `quadprog` `'interior-point-convex'` algorithm and the `lsqlin` `'interior-point'` algorithm: `'sparse'` or `'dense'`. The previous way to choose the linear algebra type was implicitly, by the type of the quadratic matrix (`H` for `quadprog`, `C` for `lsqlin`). The default behavior retains this implicit choice as the `'auto'` setting. For details, see the `options` entries on the function reference pages.

The returned `output` structure for these solvers now has a `'linearsolver'` field, with value `'sparse'` or `'dense'` depending on the linear algebra type used.

## Functionality Being Removed or Changed

**solve(prob,solver), solve(prob,options), and solve(prob,solver,options) syntaxes have been removed**
*Errors*

To choose options or the underlying solver for `solve`, use name-value pairs. For example,

```
sol = solve(prob,'options',opts,'solver','quadprog');
```

The previous syntaxes were not as flexible, standard, or extensible as name-value pairs.

**Complex Numbers Disallowed in Nonlinear Least-Squares Problems with Bounds**
*Behavior change*

For problems containing finite bounds, both `lsqcurvefit` and `lsqnonlin` now disallow steps leading to complex values.

Previously, the solvers would accept complex steps, and could return mysterious errors or spurious solutions. To use both bounds and complex values, split the problem into real and imaginary parts as described in Fit a Model to Complex-Valued Data.

# R2018a

## intlinprog Performance: Solve problems faster with new branching methods

`intlinprog` has new branching methods that can lead to faster solutions and faster acquisition of integer-feasible points.

The new branching methods are the `'strongpscost'` and `'reliability'` settings of the `BranchRule` option. See Branch and Bound.

Because `solve` internally calls `intlinprog` for integer-constrained problems, it can also use these new branching methods.

## solve Initial Point: Warm-start branch-and-bound when solving mixed-integer linear programs in the problem-based workflow

The function `solve` now accepts an optional initial point.

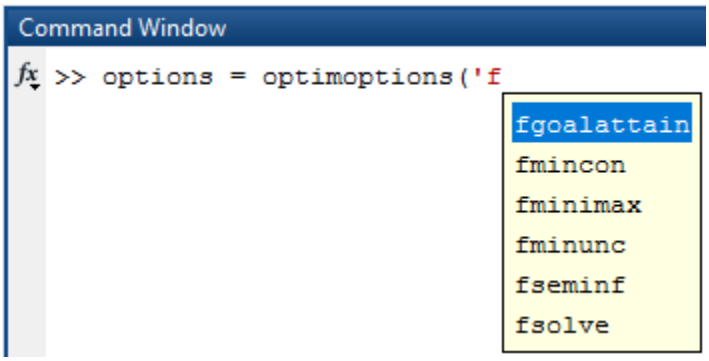
## Compatibility Considerations

The syntax of `solve` has changed. Pass the `solver` and `options` arguments in name-value pairs. Pass the optional initial point `x0` as the second argument. See the `solve` function reference page.

## Automatic Code Suggestions and Completions: Specify options and arguments by making selections from a list

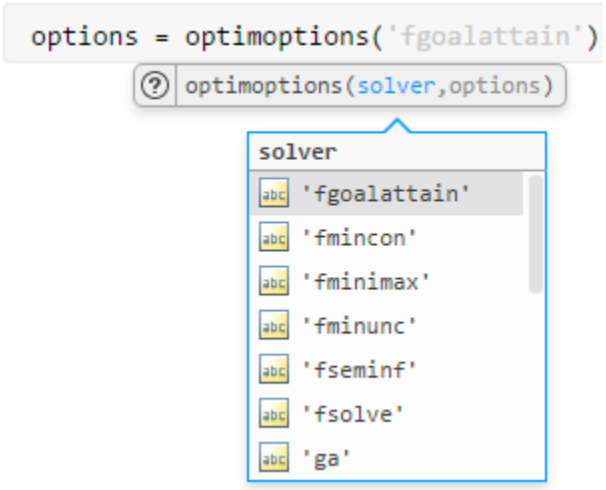
The `optimproblem`, `optimconstr`, `optimexpr`, `resetoptions`, and `optimoptions` functions present you with a list of choices for nonnumeric entries. To get the list at the command line or MATLAB Editor, enter a single or double quote and some letters, and then press **Tab**.

```
options = optimoptions('fTab
```



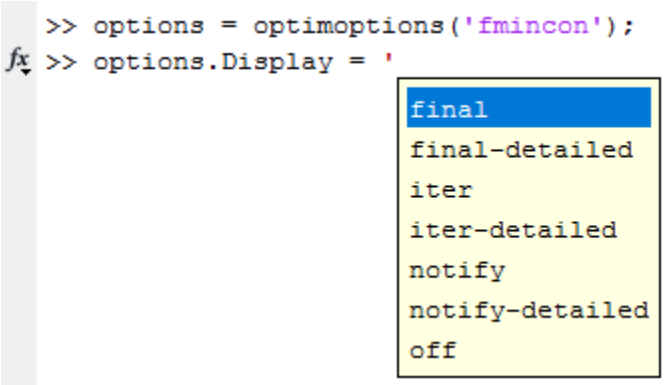

```
options = optimoptions('fminunc','DTab

options = optimoptions('fminunc','Display'
```

The Live Editor presents these choices without requiring you to press **Tab**.




You can also press **Tab** to display a list of options when using dot notation.

```
options = optimoptions('fmincon');
options.Display = 'Tab
```




## findindex: Find numeric equivalents of named index variables

To ease the creation of initial points or the interpretation of solutions when a problem has named index variables, use `findindex`. For details, see Named Index for Optimization Variables and `findindex`.

## Problem-Based Approach: Streamlined problems

The infrastructure for generating optimization expressions and constraints has changed. Generally, the resulting optimization objects are now more compact and run slightly faster.

## Compatibility Considerations

If you load an expression, constraint, or problem object saved in R2017b format, the loaded object does not inherit the new capabilities. In this case, `solve` runs the problem the same way as in R2017b, which can be slightly slower than if you create the problem in R2018a. To obtain the latest version, recreate the object using the commands that previously created the object.

Generally, you cannot load an optimization expression or problem created in R2018a into MATLAB R2017b.

## intlinprog Options: Default value changes

The `intlinprog` `ObjectiveImprovementThreshold` option now has a default value of `0`. Previously, the default was `1e-4`.

The `LPMaxIterations` option now has a default value of `'max(30000,10*(numberOfEqualities +numberOfInequalities+numberOfVariables))'`. Previously, the default was `30000`.

## Compatibility Considerations

Although the change to `ObjectiveImprovementThreshold` can provide a more accurate solution and more integer feasible points, when using the new default value, some problems run slower or do not converge. To solve a problem as before, set the `ObjectiveImprovementThreshold` to `1e-4` using `optimoptions`.

If your code assumes that the default value of `LPMaxIterations` is numeric, change the code to accept a character vector.

## Functionality Being Removed or Changed

The following functionality has changed.

| Functionality | Result | Use Instead | Compatibility Considerations |
|---|---|---|---|
| `solve(prob,solver)`, `solve(prob,options)`, or `solve(prob,solver,options)` | Warns | `solve(prob,'solver',solver)`, `solve(prob,'options',options)`, or `solve(prob,'solver',solver,'options',options)` | Replace all instances of the `solver` and `options` arguments with corresponding name-value pair arguments. |
| `str2sub` | Errors | `findindex` | Replace all instances of `str2sub` with `findindex`. |

# R2017b

**Version: 8.0**

**New Features**

**Bug Fixes**

## Optimization Modeling: Use variable expressions to represent linear or integer constraints and objectives

Solve linear programming problems and mixed-integer linear programming problems using a problem-based, variable approach.

- Create an optimization problem using `optimproblem`.
- Create optimization variables using `optimvar`.
- Specify the objective by creating an optimization expression as a linear combination of optimization variables.
- Create constraints as linear expressions in optimization variables with a comparison operator: `<=`, `==`, or `>=`.
- Solve the problem using `solve`.

For the workflow steps, see Problem-Based Workflow. For details and examples, see Problem-Based Optimization.

## Optimization Modeling: Create a collection of constraints with a single statement

For structured constraints, you can create multiple constraints in an optimization problem by using a single vectorized statement. For example, to represent the constraints that each row of a matrix `x` sums to one:

```
rowcons = sum(x,2) == 1;
```

For details, see Expressions for Constraints.

## Optimization Solving: Solve linear and mixed-integer linear problems with an automatically selected solver

The `solve` function automatically chooses the appropriate underlying solver, either `linprog` or `intlinprog`, to solve linear problems with or without integer constraints. For details, see the `solve` function reference page.

## Optimization Modeling Examples: Learn how to specify a model with examples from finance, supply chain, energy production, and more

The linear programming and mixed-integer linear programming featured examples show the problem-based approach Optimization Toolbox Examples. For examples using the previous solver-based approach, see Solver-Based Optimization.

## intlinprog Initial Point: Warm start branch-and-bound

You can specify an integer-feasible initial point for `intlinprog`. Typically, when you specify an integer-feasible point, `intlinprog` runs somewhat faster. For details, see the function reference page.

## Compatibility Considerations

For compatibility with previously-accepted syntax, the solver runs as before when you leave out the `x0` argument and include an `options` argument. In other words, the following syntaxes are equivalent when you have no initial point, meaning `x0 = []`:

```
x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,x0,options)
% and
x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,options)
```

## Functionality Being Removed or Changed

The following functions and algorithms have been removed.

| Functionality | Result | Use Instead | Compatibility Considerations |
|---|---|---|---|
| `color` removed | Errors | There is no replacement | To avoid errors,ensure that your code does not call `color`. |
| `fzmult` removed | Errors | There is no replacement | To avoid errors, ensure that your code does not call `fzmult`. |
| `gangstr` removed | Errors | There is no replacement | To avoid errors, ensure that your code does not call `gangstr`. |
| `linprog` `'simplex'` and `'active-set'` algorithms removed | Errors | `'dual-simplex'`, `'interior-point'`, or `'interior-point-legacy'` | Set the `Algorithm` option using `optimoptions` |
| `lsqlin` `'active-set'` algorithm removed | Errors | `'trust-region-reflective'` or `'interior-point'` | Set the `Algorithm` option using `optimoptions` |

# R2017a

## Interior-Point Algorithms: Solve dense quadratic and linear least-squares problems faster

The `quadprog 'interior-point-convex'` algorithm now has better performance on problems with a dense quadratic matrix $H$ in the minimization $\min_x \frac{1}{2}x^T Hx + f^T x$. For details, see Quadratic Programming Algorithms.

The `lsqlin 'interior-point'` algorithm internally calls the `quadprog 'interior-point-convex'` algorithm, as described in Interior-Point Linear Least Squares. Therefore, the `lsqlin 'interior-point'` algorithm now has better performance on problems with a dense matrix $C$ in the minimization $\min_x \frac{1}{2}\|C \cdot x - d\|_2^2$.

`lsqlin` now uses the `StepTolerance` option. For both solvers, there are new exit flag values of `2` (step size less than `StepTolerance`, constraints satisfied) and –2 (step size less than `StepTolerance`, but constraints are not satisfied). The default value of `StepTolerance` is `1e-12`, which is less than the previous default value of `1e-8` for `quadprog`.

## Compatibility Considerations

If you pass a dense quadratic matrix $H$ to `quadprog` or a dense matrix $C$ to `lsqlin`, the solution can be slightly different than before. To get the same solution, pass the matrix in `sparse` form.

To use the same default `quadprog` `StepTolerance` option as before, set `StepTolerance` to `1e-8` using `optimoptions`.

## intlinprog Heuristics: Find more integer-feasible points using new, simplified options

There are three new `intlinprog` `Heuristics` options. These options enable `intlinprog` to solve some previously-intractable problems, and to solve some problems faster. The new options are:

- `'basic'` (default)
- `'intermediate'`
- `'advanced'`

Select `Heuristics` using `optimoptions`.

`intlinprog` now uses more diving heuristics For details, see Heuristics for Finding Feasible Solutions.

## Compatibility Considerations

The default value of the `Heuristics` option is now `'basic'`. To have `intlinprog` use the previous default heuristic, set the `Heuristics` option to `'rss'`.

Heuristics that include diving cause `intlinprog` iterations to differ from previous releases. Generally, the new heuristics allow `intlinprog` to solve more problems than before. However, `intlinprog` performance can vary from previous releases.

## fmincon 'sqp' Algorithm: Use less memory

The `fmincon 'sqp'` algorithm now uses less memory. The algorithm runs faster on many problems.

## fminunc, linprog, and lsqlin: Default algorithm changes

The `fminunc`, `linprog`, and `lsqlin` solvers have new default algorithms:

| Solver | New Default Algorithm | Previous Default Algorithm |
|---|---|---|
| `fminunc` | `'quasi-newton'` | `'trust-region'` |
| `linprog` | `'dual-simplex'` | `'interior-point-legacy'` |
| `lsqlin` | `'interior-point'` | `'trust-region-reflective'` |

## Compatibility Considerations

If you do not set an algorithm, these solvers run different algorithms than before. Solutions can have slight numerical differences from the previous versions. To use a previous default algorithm, use `optimoptions` to set the `'Algorithm'` option appropriately.

## String Arguments: Solvers accept strings

You can use a `string` wherever you would previously have used a character vector in setting options or passing arguments to solvers. For example,

```
solver = string('fminunc');
name1 = string('SpecifyObjectiveGradient');
value1 = true;
name2 = string('Algorithm');
value2 = string('trust-region');
options = optimoptions(solver,name1,value1,name2,value2);
```

## Dual-Simplex Algorithm Updates: Increased robustness in intlinprog and linprog

The `'dual-simplex'` algorithm in `intlinprog` and `linprog` was updated. The algorithm now solves more problems than before.

## Compatibility Considerations

The `intlinprog` branch-and-bound iterations now take different steps than before. This change can lead to longer solution times.

## intlinprog 'RootLPMaxIterations' Option: New default value

The `intlinprog 'RootLPMaxIterations'` option has a new default value of `max(3e4, 10*(numberOfEqualities + numberOfInequalities + numberOfVariables))`. In this expression, `numberOfEqualities` means the number of rows of `Aeq`, `numberOfInequalities` means the number of rows of `A`, and `numberOfVariables` means the number of elements of `f`. This

new default value enables `intlinprog` to solve more problems than before, when the default value was `3e4`.

## Compatibility Considerations

To run `intlinprog` as before, set the `'RootLPMaxIterations'` option to `3e4` using `optimoptions`.

## Functionality Being Removed or Changed

The following functions will be removed in a future release.

| Functionality | Result | Use Instead | Compatibility Considerations |
|---|---|---|---|
| `color` | Still runs | There is no replacement | To avoid future errors, ensure that your code does not call `color`. |
| `fzmult` | Still runs | There is no replacement | To avoid future errors, ensure that your code does not call `fzmult`. |
| `gangstr` | Still runs | There is no replacement | To avoid future errors, ensure that your code does not call `gangstr`. |

# R2016b

## fmincon 'sqp' Algorithm: Solve problems more quickly

The `fmincon` `'sqp'` algorithm was rewritten. The new version is generally faster and more memory-efficient than before. Specify the new algorithm by setting the `Algorithm` option to `'sqp'`, and specify the previous algorithm as `'sqp-legacy'`.

## Compatibility Considerations

The `'sqp'` and `'sqp-legacy'` algorithms are similar, but have slight numerical differences that can lead to different results.

If you load a problem that was saved with a MATLAB version prior to R2016b that had the `fmincon` `Algorithm` option set to `'sqp'`, then MATLAB converts the problem options to the `'sqp-legacy'` algorithm.

The Optimization app cannot run or save the `'sqp-legacy'` algorithm. It converts any such problem into one that uses the new `'sqp'` algorithm.

You cannot choose the `'sqp-legacy'` algorithm using `optimset`. Instead, use `optimoptions`.

## intlinprog Heuristics: Select from a larger set of heuristics for finding integer-feasible points

There are more options for `intlinprog` `Heuristics`. The new options generally have `-diving` appended to the previous names, and are:

- `'diving'`
- `'round-diving'`
- `'rss-diving'`
- `'rins-diving'`

The new `-diving` options can enable `intlinprog` to solve previously intractable problems. Select `Heuristics` using `optimoptions`. For details, see Heuristics for Finding Feasible Solutions.

## intlinprog: Revised branch-and-bound algorithm

`intlinprog` has a new branch-and-bound implementation. The intent is to have similar or better performance than the previous version, and to be more memory-efficient.

## Compatibility Considerations

The new algorithm can take different iterative steps, and can find different integer-feasible points than before.

## quadprog: 'active-set' algorithm removed

The `quadprog` `'active-set'` algorithm has been removed.

## Compatibility Considerations

If you load a problem that was saved with a previous toolbox version, and that has the `Algorithm` option `'active-set'`, then `optimoptions` changes the option to the default `'interior-point-convex'` algorithm.

## linprog: 'active-set' and 'simplex' algorithms removed, syntax update, and default algorithm will change

The `linprog` `'active-set'` and `'simplex'` algorithms have been removed. Additionally, you can no longer choose the `linprog` algorithm using the `LargeScale` option.

Previously, `linprog` accepted an initial point argument `x0` for the `'active-set'` algorithm. Now that this algorithm is removed, you no longer need to pass `x0` to `linprog`. This means that the following syntax is valid:

```
x = linprog(f,A,b,Aeq,beq,lb,ub,options)
```

If you do not set an algorithm, `linprog` warns that, in a future release, the default will change from `'interior-point-legacy'` to `'dual-simplex'`.

## Compatibility Considerations

If you load a problem that was saved with a previous toolbox version, and that has the `Algorithm` option `'active-set'` or `'simplex'`, then `optimoptions` changes the option to the default `'interior-point-legacy'` algorithm.

Choose the `linprog` algorithm by using `optimoptions` to set the `Algorithm` option.

For compatibility, `linprog` accepts the former syntax that includes `x0`, but `linprog` doesn't use `x0`:

```
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

## fsolve: 'trust-region-reflective' algorithm renamed 'trust-region'

The former `fsolve` `'trust-region-reflective'` algorithm is now named `'trust-region'`.

## Compatibility Considerations

When you set the `Algorithm` option using `optimoptions`, you can use either the former name `'trust-region-reflective'` or the current name `'trust-region'`. However, `optimoptions` displays the resulting option name as `'trust-region'` in either case.

To set the algorithm using `optimset`, use the former name `'trust-region-reflective'`.

If you export options containing the `'trust-region'` algorithm from the Optimization app, the exported options cannot be used in toolbox versions prior to R2016a.

## lsqlin: 'active-set' algorithm will be removed, default algorithm will change

The `lsqlin` `'active-set'` algorithm now warns that it will be removed in a future release.

When you do not set an `Algorithm` option, `lsqlin` warns that the default algorithm will change to `'interior-point'` in a future release.

### Compatibility Considerations

To avoid these warnings, use `optimoptions` to set the `Algorithm` option to `'trust-region-reflective'` or `'interior-point'`.

## fminunc: InitialHessMatrix and InitialHessType options removed

The `fminunc` `InitialHessMatrix` and `InitialHessType` options have been removed.

### Compatibility Considerations

To avoid errors, remove any reference to these options from your `optimset` or `optimoptions` calls.

## Featured example update

The `bandem` function, which used to bring up an app showing how various algorithms perform, has been replaced by an updated example, Banana Function Minimization.

# R2016a

## Renamed Options: Use more expressive and consistent names for options

Many options have new names. The old names continue to work, but do not appear in documented examples. The renaming gives a more consistent set of option names that match those in Global Optimization Toolbox.

All legacy option names continue to work in both `optimoptions` and `optimset`.

For option name change details, see Current and Legacy Option Name Tables.

## Compatibility Considerations

`optimset` uses only the legacy option names, while `optimoptions` uses either legacy names or current names. However, `optimoptions` displays only current names, so if you set a legacy name, it displays the equivalent current name. See Current and Legacy Option Name Tables.

`optimoptions` no longer displays some options. See View Options.

Some option values differ when using current names. For example, any previous setting that was `'on'` or `'off'` is now `true` or `false`.

## Parallel Computation: Accelerate fminunc, fsolve, lsqcurvefit, and lsqnonlin functions (using Parallel Computing Toolbox)

More nonlinear solvers can estimate gradients or Jacobians by parallel finite differences:

- `fminunc`
- `fsolve`
- `lsqcurvefit`
- `lsqnonlin`

To enable parallel finite differences, set the `UseParallel` option to `true`. For details, see Parallel Computing.

## Option Changes: Distinguish between function tolerance and optimality tolerance, specify Hessians differently, more

- Previously, the `TolFun` tolerance applied to both the change in function value and to the size of the first-order optimality measure. This role is now explicitly split into two new tolerances:
  - `FunctionTolerance` — Applies to changes in objective function value
  - `OptimalityTolerance` — Applies to first-order optimality measure
- The way to set Hessians for `fminunc` and `fmincon` has changed. For details, see Including Hessians.
- Several solvers gained a `SubproblemAlgorithm` option for their `'trust-region'` or `'trust-region-reflective'` algorithm:
  - `fminunc`

  - `fsolve`
  - `lsqcurvefit`
  - `lsqnonlin`
- The former `Jacobian` option in `fsolve`, `lsqcurvefit`, and `lsqnonlin` is now the `SpecifyObjectiveGradient` option.

## Compatibility Considerations

The Optimization app imports and exports only one option related to the former `TolFun` tolerance. It displays this option as **Optimality tolerance**, and uses it as the `OptimalityTolerance` option. You cannot import, export, or change the `FunctionTolerance` option in the Optimization app for Optimization Toolbox solvers. When you run problems in the Optimization app, `FunctionTolerance` has its default value.

However, Global Optimization Toolbox solvers do not have an `OptimalityTolerance` option. Those solvers can import, export, and set the `FunctionTolerance` option in the Optimization app.

## resetoptions: Restore default option values

The `resetoptions` function resets selected `optimoptions` options to their default values. For details, see the function reference page.

## Iterative display changes in linprog and quadprog

Iterative display has changed for the `linprog` `'dual-simplex'` and `'interior-point'` algorithms, and for the `quadprog` `'interior-point-convex'` algorithm. The changes give clearer and more consistent information. For iterative display details, see linprog and quadprog.

## fsolve 'trust-region-dogleg' algorithm: Improved robustness

The default `fsolve` `'trust-region-dogleg'` algorithm is now more robust to function evaluation failures during finite difference estimation of the Jacobian.

# R2015b

## Additional interior-point linear programming algorithm with improved performance and robustness

The `linprog` solver has a new algorithm named `'interior-point'`. The previous algorithm named `'interior-point'` is now named `'interior-point-legacy'`. Usually, the new `'interior-point'` algorithm is faster and more robust than the `'interior-point-legacy'` algorithm.

## Compatibility Considerations

The default `linprog` algorithm is now named `'interior-point-legacy'`. If your code explicitly chooses the `'interior-point'` algorithm, `linprog` runs the new interior-point algorithm, and so can have different behavior than before.

## Mathematical Programming System (MPS) file reader for importing linear programming and mixed-integer linear programming problems

The `mpsread` function reads files in the MPS format that specify linear programming problems or mixed-integer linear programming problems. For details, see the function reference page.

## Updated output structure in several solvers

The `fmincon`, `fsolve`, `lsqcurvefit`, and `lsqnonlin` `'trust-region-reflective'` algorithms now return a `stepsize` field in their `output` structures. Similarly, the `fminunc` `'trust-region'` algorithm now returns a `stepsize` field in its `output` structure.

The `fmincon` `'sqp'` algorithm and `fminunc` `'quasi-newton'` algorithm now return a `stepsize` field that contains the size of the final step, and a `lssteplength` field that contains the relative step compared to a line-search prediction.

## Compatibility Considerations

The meaning of the `stepsize` field has changed for the `output` structure of the `fmincon` `'sqp'` algorithm and `fminunc` `'quasi-newton'` algorithm. The former `stepsize` field, which measured the relative step compared to a line-search prediction, is now returned in the `lssteplength` field.

## Optimization app will be removed in a future release

The Optimization app now warns that it will be removed in a future release.

## Compatibility Considerations

Set and examine options using `optimoptions`, which streamlines viewing Optimization Toolbox options.

## Linear programming example

There is a new featured example of linear programming, Maximize Long-Term Investments Using Linear Programming.

# R2015a

## Improved performance and robustness of intlinprog primal-simplex algorithm

The `'primal-simplex'` algorithm of the `intlinprog` solver can solve more problems than before, and has better performance on large problems. Select the algorithm in the `RootLPAlgorithm` option.

## Compatibility Considerations

`intlinprog` iterations can differ from previous versions. This holds even when you select the `'dual-simplex'` algorithm, because `intlinprog` uses the `'primal-simplex'` algorithm for some calculations in any case.

## linprog dual-simplex algorithm returns more information

The `linprog` `'dual-simplex'` algorithm now returns a Lagrange multiplier structure and an `output` structure containing the first-order optimality measure.

## quadprog active-set algorithm will be removed

The `quadprog` `'active-set'` algorithm now warns that it will be removed in a future release.

## Compatibility Considerations

To avoid this warning, use `optimoptions` to set the `Algorithm` option to `'interior-point-convex'` or `'trust-region-reflective'`. Or simply do not set the `Algorithm` option; `quadprog` defaults to the `'interior-point-convex'` algorithm.

## fmincon allows problems without constraints

`fmincon` no longer throws an error if you run it on a problem without constraints. This change can make it easier for you to see the effect of constraints by running a problem both with and without constraints.

To run `fmincon` without constraints, you no longer need to set an artificial constraint, such as `lb = -Inf`, but doing so will not matter.

## Compatibility Considerations

Any code that checks for this error will have different behavior than before.

## Least-squares solvers allow equal upper and lower bounds

The `lsqnonlin` and `lsqcurvefit` solvers now allow you to fix variables by specifying equal upper and lower bounds. For example, lower bound `lb = [0,0]` and upper bound `ub = [Inf,0]` fix the solution `x` to have `x(2) = 0`.

## quadprog allows zero or empty quadratic term

Previously, if you gave a zero or empty matrix as the `H` input, the `quadprog` solver would switch to `linprog`. Now `quadprog` solves the problem.

## Changes to algorithm field of output structure

The `algorithm` field of the `output` structure of several solvers has changed. Now, all `output.algorithm` strings are the same as those you set in the `Algorithm` name-value pair. For example, the `output.algorithm` field of the `fmincon` `'active-set'` algorithm used to be `'medium-scale: SQP, Quasi-Newton, line-search'`. Now it is `'active-set'`.

The affected solvers are `fgoalattain`, `fminimax`, `fmincon`, `fseminf`, `fsolve`, `linprog`, `lsqcurvefit`, and `lsqnonlin`.

## Compatibility Considerations

Any code that uses the string in `output.algorithm` can have different behavior than before.

## Mixed-integer quadratic programming example

There is a new featured example of mixed-integer quadratic programming, Mixed-Integer Quadratic Programming Portfolio Optimization.

# R2014b

## Dual-simplex algorithm in linprog linear programming solver

`linprog` has a new algorithm option named `'dual-simplex'`. The algorithm saves memory when you specify sparse constraint matrices. For details, see the function reference page.

Currently, `'dual-simplex'` returns an empty Lagrange multiplier structure `lambda` and `firstorderopt` field in the `output` structure.

## Interior-point algorithm in lsqlin linear least-squares solver

`lsqlin` has a new algorithm option named `'interior-point'`. The algorithm handles all types of constraints and saves memory when you specify sparse constraint matrices. For details, see the function reference page.

## Plot functions and output functions for monitoring progress of intlinprog solver

`intlinprog` accepts `OutputFcn` and `PlotFcns` options. For details, see intlinprog Output Functions and Plot Functions.

There is a built-in output function that collects all integer feasible solutions that `intlinprog` encounters. Choose this output function by using `optimoptions` to set the `OutputFcn` option to `@savemilpsolutions`.

There is a built-in plot function that shows details of the solver iterations. Choose this plot function by using `optimoptions` to set the `PlotFcns` option to `@optimplotmilp`.

## Levenberg-Marquardt InitDamping option

The `fsolve`, `lsqcurvefit`, and `lsqnonlin` solvers can use the `'levenberg-marquardt'` algorithm. To initialize the Levenberg-Marquardt parameter differently than its default, set the new `InitDamping` option using `optimoptions`.

## Compatibility Considerations

Previously, to initialize the Levenberg-Marquardt parameter you would pass a cell array in the `Algorithm` option when using `optimset`, such as

```
options = optimset('Algorithm',{'levenberg-marquardt',0.1});
```

That method still works. But when using `optimoptions`, you cannot set the parameter in the `Algorithm` option, and instead must use the `InitDamping` option, such as

```
options = optimoptions(@fsolve,'Algorithm','levenberg-marquardt','InitDamping',0.1);
```

## Changing default algorithm for fminunc

The `fminunc` default algorithm will change to `'quasi-newton'` in a future release.

## Compatibility Considerations

`fminunc` now warns when you run it in cases where the default behavior will change. For details, see the `fminunc` function reference page or fminunc Algorithms.

## Mixed-integer linear programming example

There is a new featured example of mixed-integer linear programming, Optimal Dispatch of Power Generators.

## Two linprog algorithms will be removed in the future

The `linprog` `'active-set'` and `'simplex'` algorithms warn that they will be removed in a future release.

## Compatibility Considerations

To avoid this warning, choose the `'interior-point'` or `'dual-simplex'` algorithms using `optimoptions`.

## Two fminunc options will be removed in the future

`fminunc` now warns that the `InitialHessMatrix` and `InitialHessType` options will be removed in a future release.

## Compatibility Considerations

To avoid these warnings, do not set values for these options. The Optimization app no longer has these options.

## bintprog removed

The `bintprog` function has been removed.

## Compatibility Considerations

To update code to use `intlinprog` instead of `bintprog`, see Tips.

## ktrlink removed

The `ktrlink` interface to the KNITRO® third-party solver has been removed.

## Compatibility Considerations

For a KNITRO interface, contact Ziena Optimization: `www.ziena.com`.

# R2014a

## Mixed-integer linear programming solver

The `intlinprog` function solves mixed-integer linear programming problems. For more information, see the reference page and the documentation.

There are several new featured examples of mixed-integer linear programming:

- Factory, Warehouse, Sales Allocation Model
- Travelling Salesman Problem
- Solve Sudoku Puzzles Via Integer Programming

Currently, you cannot run `intlinprog` in the Optimization app.

## New default algorithms in fmincon and quadprog

When you do not specify an algorithm, `fmincon` and `quadprog` default to different algorithms than before. Usually, the new algorithms are faster and more robust than the algorithms they replace.

| Solver | Previous Default Algorithm | New Default Algorithm |
|---|---|---|
| `fmincon` | `'trust-region-reflective'` | `'interior-point'` |
| `quadprog` | `'trust-region-reflective'` | `'interior-point-convex'` |

## Compatibility Considerations

Solvers can produce different results than before. To reproduce previous results, set the `Algorithm` option with `optimoptions`.

```
options = optimoptions('fmincon','Algorithm','trust-region-reflective');
% or
options = optimoptions('quadprog','Algorithm','trust-region-reflective');
```

Be sure to pass `options` in your function call.

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nlcon,options)
% or
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
```

## Nonlinear least-squares tweaks

The `lsqcurvefit` and `lsqnonlin` solvers have slightly different behavior than before when the internally calculated trust-region radius gets large in the `'trust-region-reflective'` algorithm, or the Levenberg-Marquardt parameter gets small in the `'levenberg-marquardt'` algorithm. There are no longer any arbitrary limits on these parameters. `fsolve` has the same change to its `'trust-region-reflective'` and `'levenberg-marquardt'` algorithms.

## Parallel option change

The `UseParallel` option now accepts the values `true` and `false`. The option also accepts the former values `'always'` and `'never'`, and scalar values `1` and `0`.

The affected solvers are `fmincon`, `fgoalattain`, and `fminimax`.

## ktrlink default math library change

The `ktrlink` function uses a different default math library within KNITRO than before. This change can enable `ktrlink` to run on more hardware versions.

## Compatibility Considerations

With the new default math library, `ktrlink` runs slower than before. To use the previous math library, set the KNITRO format option `blasoption` to 1. See Setting Options.

## bintprog will be removed in the future

`intlinprog` solves more problems than `bintprog`, and has better performance. So `bintprog` will be removed in a future release.

To update your existing `bintprog` code to use `intlinprog`, see Tips.

## ktrlink will be removed in the future

`ktrlink` will be removed in a future release. For an updated KNITRO interface, contact Ziena Optimization: `www.ziena.com`.

# R2013b

## Solvers that check initial point more carefully

All nonlinear solvers now check whether derivatives of the objective and nonlinear constraint functions are well defined at the initial point, usually called `x0`. (This is in addition to the existing checks that the functions are well defined at `x0`.) Well defined means the value of each derivative is not `NaN`, `Inf`, or complex (nonlinear least-squares solvers allow complex values). Derivatives include both gradients and Jacobians. See Including Derivatives and Writing Vector and Matrix Objective Functions.

When the objective or nonlinear constraint functions do not include a derivative, solvers approximate derivatives by finite differences. This means that the functions must be well defined for points in a small neighborhood of `x0`.

If any derivative is not well defined at `x0`, the solver stops with an error, and does not attempt to find a solution.

In all tested cases, this behavior led to a clearer exit condition.

## Compatibility Considerations

It is conceivable that a problem that previously ran to completion will now exit without completion. This can occur in the rare case when a derivative did not exist for a function at the initial point, but the solver was able to step to a point where the derivatives were well defined.

# R2013a

## optimoptions function for setting options with compact and comprehensive display

The new `optimoptions` function creates and modifies options for all solvers except the base MATLAB solvers `fminbnd`, `fminsearch`, `fzero`, and `lsqnonneg`. Continue to use `optimset` for those functions.

`optimoptions` organizes options by solver, with a more focused and comprehensive display than `optimset`:

- Creates and modifies only the options that apply to a solver
- Shows your option choices and default values for a specific solver/algorithm
- Displays links for more information on solver options and other available solver algorithms

For details, see the `optimoptions` page, or Set Options.

## Compatibility Considerations

If you export options or a problem from the Optimization Tool, the options are an object as created by `optimoptions`. Therefore, earlier software versions cannot import the options or problem. This consideration does not apply to the base MATLAB solvers, which continue to encapsulate options as a structure.

## Algorithm option replaces LargeScale option

Use the `Algorithm` option to select the algorithm in all solvers that have multiple algorithms. In a future release, solvers will no longer use the `LargeScale` option. The `linprog` `Simplex` option will also change to the `Algorithm` option.

## Compatibility Considerations

Update options as follows.

| Solver | Previous Setting | Current Setting |
|---|---|---|
| `fminunc` | `'LargeScale' = 'on'` | `'Algorithm' = 'trust-region'` |
| | `'LargeScale' = 'off'` | `'Algorithm' = 'quasi-newton'` |
| `linprog` | `'LargeScale' = 'on'` | `'Algorithm' = 'interior-point'` |
| | `'LargeScale' = 'off'`, `'Simplex' = 'off'` or unset | `'Algorithm' = 'active-set'` |
| | `'LargeScale' = 'off'`, `'Simplex' = 'on'` | `'Algorithm' = 'simplex'` |
| `lsqlin` | `'LargeScale' = 'on'` | `'Algorithm' = 'trust-region-reflective'` |
| | `'LargeScale' = 'off'` | `'Algorithm' = 'active-set'` |

## ktrlink supports KNITRO 8.1

The `ktrlink` function now supports KNITRO version 8.1. For details, see ktrlink: An Interface to KNITRO Libraries.

# R2012b

## Changing default algorithms for fmincon and quadprog

The `fmincon` and `quadprog` default algorithms will change in a future release.

- The default `fmincon` algorithm will become `'interior-point'`.
- The default `quadprog` algorithm will become `'interior-point-convex'`.

## Compatibility Considerations

These solvers now warn when you run them in cases where the default behavior will change. For example, they warn when:

- You do not set the `Algorithm` option.
- You set incompatible `LargeScale` and `Algorithm` options.

To avoid these warnings:

- Do not set the `LargeScale` option.
- Set the `Algorithm` option appropriately.

For details, see the `fmincon` and `quadprog` function reference pages or Choosing the Algorithm.

## ktrlink supports KNITRO 8

The `ktrlink` function now supports KNITRO version 8. For details, see ktrlink: An Interface to KNITRO Libraries.

# R2012a

## Enhanced Robustness in fminunc

The `fminunc` medium-scale algorithm now attempts to recover from failures when evaluating the objective function during iteration steps, or during gradient estimation. Failure means the objective function returns `NaN`, a complex value, or `Inf`. If there is such a failure, the algorithm attempts to take different steps.

As part of robustness, the `fminunc` medium-scale algorithm now uses the `ObjectiveLimit` tolerance.

## Compatibility Considerations

When objective function values drop below `ObjectiveLimit` (default value: `-1e20`), iterations end with a `-3` exit flag. Use `optimset` to change the value of `ObjectiveLimit`. Set `ObjectiveLimit` to `-Inf` to disable this tolerance.

## FinDiffRelStep Option in Optimization Tool

The `FinDiffRelStep` option for choosing relative finite difference step sizes is now available in the Optimization Tool, in the **Approximated derivatives** pane. This option lets you tune the gradient estimation step in most solvers.

## Levenberg-Marquardt Algorithm Tweak

The `fsolve`, `lsqcurvefit`, and `lsqnonlin` solvers no longer use the magnitude of the Levenberg-Marquardt regularization parameter as a stopping criterion, so they no longer return an exit flag of `-3` when using the `levenberg-marquardt` algorithm. Instead, they use the `TolX` tolerance in all internal calculations.

## Compatibility Considerations

The solvers now stop with exit flag `2` in most situations where previously they stopped with exit flag `-3`.

## fmincon sqp Algorithm Tweak

The `fmincon` `sqp` algorithm calculates its Lagrange multiplier estimates somewhat differently than before.

## Compatibility Considerations

The `fmincon` `sqp` algorithm can give slightly different results than before.

# R2011b

## Derivative Estimate Changes

- The `fsolve`, `lsqcurvefit`, and `lsqnonlin` solvers now accept the `FinDiffType` option. Set `FinDiffType` to `'central'` with `optimset` to enable derivative estimation by central finite differences. Central finite differences are more accurate, but take more time than the default `'forward'` finite differences.
- `fsolve`, `lsqcurvefit`, and `lsqnonlin` now use the `TypicalX` option when estimating dense Jacobians via finite differences. In previous releases, these solvers used `TypicalX` only when checking derivatives.
- For algorithms that obey bounds, finite difference steps for derivative estimation now stay within any bounds you set for the decision variables. See Iterations Can Violate Constraints.
- The new `FinDiffRelStep` option allows you to set a vector of finite difference step sizes to better handle problems whose components have different scales. Use `FinDiffRelStep` at the command line for any solver that uses finite differences. For details, see `FinDiffRelStep` in Options Structure.

## Gauss-Newton Algorithm Removed

The `fsolve`, `lsqcurvefit`, and `lsqnonlin` functions no longer use the Gauss-Newton algorithm.

## Compatibility Considerations

The previous way of selecting the Gauss-Newton algorithm was to set the `LargeScale` option to `'off'`, and in:

- `fsolve` — set the `NonlEqnAlgorithm` option to `'gn'`.
- `lsqcurvefit` or `lsqnonlin` — set the `LevenbergMarquardt` option to `'off'`.

To select an algorithm, use `optimset` to set the `Algorithm` option:

- `fsolve` — `trust-region-dogleg`, `trust-region-reflective`, or `levenberg-marquardt`
- `lsqcurvefit` or `lsqnonlin` — `trust-region-reflective` or `levenberg-marquardt`

Solvers no longer use the `LevenbergMarquardt`, `LineSearchType`, and `NonlEqnAlgorithm` options, since these options relate only to the Gauss-Newton algorithm.

## DerivativeCheck Changes

The `DerivativeCheck` option checks whether a solver's finite-difference approximations match the gradient or Jacobian functions that you supply. When a solver finds a discrepancy between the computed derivatives and their finite-difference approximations, the solver now errors. Solvers used to pause in this situation instead of erroring.

Additionally, solvers now compare derivatives at a point near the initial point `x0`, but not exactly at `x0`. Previously, solvers performed the comparison at `x0`. This change usually gives more reliable `DerivativeCheck` decisions. For details, see Checking Validity of Gradients or Jacobians.

Solvers do not include the computations for `DerivativeCheck` in the function count. See Iterations and Function Counts.

## Compatibility Considerations

Solvers compare the derivatives at a different point than before, so can change their decision on whether the derivatives match. Solvers now error instead of pause when they encounter a discrepancy.

## fmincon ScaleProblem Default Changed

The `fmincon` `interior-point` and `sqp` algorithms can use the `ScaleProblem` option. The default value of `ScaleProblem` is now `'none'` instead of `'obj-and-constr'`.

## Compatibility Considerations

Because of a bug in previous releases, when you did not provide gradients of the objective and nonlinear constraint functions, `fmincon` did not scale these functions. `fmincon` did scale linear constraints. So, if you do not provide gradients and have no linear constraints, the current `fmincon` behavior is the same as in previous releases. However, the current behavior can differ if you do provide gradients (`GradObj` or `GradConstr` is `'on'`). If you provide gradients, have no linear constraints, and want to obtain the previous behavior, set `ScaleProblem` to `'obj-and-constr'` with `optimset`.

## fsolve trust-region-dogleg Algorithm Change

The `fsolve` `trust-region-dogleg` algorithm no longer performs an internal calculation of conditioning. This change usually speeds `fsolve`.

## Compatibility Considerations

`fsolve` iterations differ from previous versions. Additionally, the solution and all associated outputs can differ from previous versions. Usually, results are numerically equivalent to previous results.

## Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Optimization Toolbox.

## Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a `try`/`catch` statement and performs an action based on a specific error identifier.

For example, the `'optim:fmincon:ConstrainedProblemsOnly'` identifier has changed to `'optimlib:fmincon:ConstrainedProblemsOnly'`. If your code checks for `'optim:fmincon:ConstrainedProblemsOnly'`, you must update it to check for `'optimlib:fmincon:ConstrainedProblemsOnly'` instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

**Tip** Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs warning free.

# R2011a

## New Quadratic Programming Algorithm

`quadprog` has a new algorithm named `'interior-point-convex'`. It has these features:

- The algorithm has fast internal linear algebra.
- The algorithm handles sparse problems.
- There is a new presolve module that can improve speed, numerical stability, and detection of infeasibility.
- The algorithm handles large convex problems, and accepts and uses sparse inputs. See Large-Scale vs. Medium-Scale Algorithms.
- The algorithm optionally gives iterative display.
- The algorithm has enhanced exit messages.

For details on the algorithm, see interior-point-convex quadprog Algorithm. For help choosing the algorithm to use, see Quadratic Programming Algorithms.

## Compatibility Considerations

You now choose the `quadprog` algorithm by using `optimset` to set the `Algorithm` option instead of the `LargeScale` option. If you don't set `Algorithm` or `LargeScale`, `quadprog` behaves as before.

`Algorithm` option choices are:

- `trust-region-reflective` (formerly `LargeScale = 'on'`), the default
- `active-set` (formerly `LargeScale = 'off'`)
- `interior-point-convex`

The previous way of choosing the `quadprog` algorithm at the command line was to set the `LargeScale` option to `'on'` or `'off'`. `quadprog` now ignores the `LargeScale` option, except when you set the inconsistent values `LargeScale = 'off'` and `Algorithm = 'trust-region-reflective'`. In this case, to avoid backward incompatibility, `quadprog` honors the `LargeScale` option, and uses the `'active-set'` algorithm.

`quadprog` now checks whether any inputs are complex, and, if so, it errors. The only exception is the `Hinfo` argument for the `HessMult` option is allowed to be complex.

## Enhanced Robustness in Nonlinear Solvers

More solvers now attempt to recover from errors in the evaluation of objective functions and nonlinear constraint functions during iteration steps, or, for some algorithms, during gradient estimation. The errors include results that are `NaN` or `Inf` for all solvers, or complex for `fmincon` and `fminunc`. If there is such an error, the algorithms attempt to take different steps. The following solvers are enhanced:

- `fmincon` `trust-region-reflective` algorithm (the `interior-point` and `sqp` algorithms already had this robustness)
- `fminunc` `LargeScale` algorithm
- `fsolve` `trust-region-reflective`, `trust-region-dogleg`, and `levenberg-marquardt` algorithms

- `lsqcurvefit` `trust-region-reflective` and `levenberg-marquardt` algorithms
- `lsqnonlin` `trust-region-reflective` and `levenberg-marquardt` algorithms

## New Defaults in DiffMinChange and DiffMaxChange Options

The `DiffMinChange` and `DiffMaxChange` options set the minimum and maximum possible step sizes for finite differences in gradient estimation. The defaults are now:

- `DiffMinChange = 0` (formerly `1e-8`)
- `DiffMaxChange = Inf` (formerly `0.1`)

Solvers have mechanisms that ensure nonzero and non-infinite step sizes, so the new defaults simply mean that the step size adjustment algorithms have fewer constraints.

The new defaults remove the previous arbitrary choices. The previous values can be inappropriate when components are too large or small in magnitude. Tests show these new defaults are good for most situations.

## Compatibility Considerations

Some solver iterations can differ from previous ones. To obtain the previous behavior:

```
options = optimset('DiffMinChange',1e-8,'DiffMaxChange',0.1);
```

## Output Structure Tweak

For the `trust-region-reflective` algorithm, the `algorithm` field of the `output` structure is now `'trust-region-reflective'`. This value differs slightly from the previous values returned by `fmincon`, `fsolve`, `lsqcurvefit`, `lsqnonlin`, and `quadprog`.

## Compatibility Considerations

To avoid errors or unexpected results, update any code that depends on the exact value of the `output.algorithm` string.

## ktrlink Compatible with KNITRO 7

`ktrlink` is compatible with KNITRO 7. For details, see ktrlink: An Interface to KNITRO Libraries, or the Ziena Optimization web site `http://www.ziena.com/`.

## New quadprog Demo

The new demo `Using Quadratic Programming on Portfolio Optimization Problems` shows how to solve portfolio optimization problems using `quadprog`.

# R2010b

**Version: 5.1**

**New Features**

**Bug Fixes**

## Enhanced fmincon Finite Difference Algorithms Add Robustness

The `fmincon` `interior-point` and `sqp` algorithms now attempt to recover from errors in the evaluation of objective functions and nonlinear constraint functions during gradient estimation. The errors include results that are `NaN`, `Inf`, or complex. If there is such an error, the finite differencing routines attempt to take different steps.

## ktrlink Available for Macintosh 64-Bit Systems

The `ktrlink` function now works with Macintosh 64-bit systems. Therefore, `ktrlink` works on the same systems as all other Optimization Toolbox functions.

## Output Structure Tweaks

All `linprog` and `quadprog` algorithms now create a `firstorderopt` field in the output structure. This field contains the value of the first-order optimality measure at the final point.

All `fmincon` and `quadprog` algorithms now create a `constrviolation` field in the output structure. This field contains the largest value of the constraint functions at the final point: bounds, linear constraints, and nonlinear constraints. (Some algorithms return the larger of the constraint functions and 0.) See Writing Constraints.

## New Video Demo on Modeling

There is a new two-part video on modeling and solving optimization problems:

- Mathematical Modeling with Optimization, Part 1
- Optimization Modeling 2: Converting to Solver Form

# R2010a

## New fmincon Algorithm

`fmincon` has a new algorithm called `SQP` for Sequential Quadratic Programming. The algorithm has the following features:

- Honors bounds at all iterations
- Attempts a different step if one leads to an objective or constraint function returning a `NaN`, `Inf`, or complex result
- Fast internal linear algebra for solving quadratic programs

Choose the algorithm at the command line by setting the `Algorithm` option to `'sqp'` with `optimset`. For more information about the algorithm, see fmincon SQP Algorithm.

## lsqnonneg No Longer Uses x0

The `lsqnonneg` solver no longer accepts a start point `x0` as an optional input.

## Compatibility Considerations

The Optimization Tool no longer has an input region for accepting a start point. If you import or run a problem that contains a start point `x0`, MATLAB issues a warning. Also, the Optimization Tool and `lsqnonneg` ignore `x0`, and instead use a start point of a vector of zeroes. If you export a problem structure from the Optimization Tool, there is no `x0` field.

# R2009b

## Enhanced Exit Messages in Selected Solvers

Enhanced, clearer exit messages in `fsolve`, `lsqnonlin`, and `lsqcurvefit`, with links for more information. For more information about the enhancements, see Exit Flags and Exit Messages.

## Compatibility Considerations

For solvers with enhanced exit messages, the content of `output.message` contains many more characters than before. User code that relies on this field might need to be modified in order to display the larger exit message satisfactorily.

## fmincon Interior-Point Algorithm Robust to Certain Errors

The `fmincon` interior-point algorithm attempts to continue when a user-supplied objective or constraint function returns `Inf`, `NaN`, or a complex result. For more information, see fmincon Interior Point Algorithm.

## Changes in quadprog

The large-scale `quadprog` algorithm now uses the `TolFun` and `MaxIter` tolerances for deciding when to end iterations when there are only linear equality constraints, instead of the `TolPCG` and `MaxPCGIter` tolerances.

The `quadprog` `output` structure now contains the `constrviolation` field, which reports the maximum constraint function at the final point.

## Compatibility Considerations

For large-scale linear equality constrained problems, the default values of the tolerances are much tighter than before, so `quadprog` can take more iterations, but the resulting solution should be more accurate.

## Changes in linprog

The large-scale interior-point algorithm of `linprog` now has a backtracking mechanism for the case of stalling, and performs LDL factorization when there is rank deficiency. For more information, see Large Scale Linear Programming.

The `linprog` `output` structure now contains the `constrviolation` field, which reports the maximum constraint function at the final point.

## Compatibility Considerations

The interior-point algorithm of `linprog` might arrive at different solutions than before, and can solve more problems than before.

## Multiobjective optimValues Changes

The `optimValues` structure, used by output functions, has two new fields to better reflect the state of multiobjective solvers:

- For `fgoalattain`, the `optimValues.attainfactor` field contains the value of $\gamma$, the attainment factor.
- For `fminimax`, the `optimValues.maxfval` field contains the value $\max_i F_i$, where $F$ is the vector of objectives.

Furthermore, the value stored in `optimValues.fval` has changed. Now `optimValues.fval` contains the vector $F$ of objective function values. For a complete description of the current `optimValues` structure, see Fields in optimValues.

## Compatibility Considerations

User code that uses the `optimValues.fval` field within an output function in `fgoalattain` and `fminimax` might need to be updated to avoid errors

# R2009a

## Parallel Gradient Estimation Available in fmincon Interior-Point Algorithm

The `fmincon` solver's `interior-point` algorithm can now compute finite differences in parallel in order to speed the estimation of gradients. For details on how to use this parallel gradient estimation, see the Parallel Computing for Optimization chapter in the User's Guide.

## Enhanced Exit Messages in Selected Solvers

Solvers print exit messages by default at the end of their runs. The exit messages are different in R2009a for several solvers, and the messages have been enhanced with new functionality. The following sections describe the new features and changes. There is more information in the Exit Flags and Exit Messages section of the User's Guide.

The following solvers have enhanced exit messages:

- `fgoalattain`
- `fmincon`
- `fminimax`
- `fminunc`
- `fseminf`

### Links to More Information Window

The enhanced exit messages include hyperlinks within their exit messages. These hyperlinks bring up a window containing further information about the terms used in the exit messages.

### Link for More Detail in Command Window

A `<stopping criteria details>` hyperlink may appear at the end of an exit message, depending on the solver and setting of the `Display` option. This link causes the solver to print more detail about the exit conditions to the MATLAB Command Window.

### New Display Option Values Control Default Detail

There are new values of the `Display` option to control whether detailed exit messages appear instead of the default (simpler) messages. The new values are:

- `'final-detailed'`
- `'iter-detailed'`
- `'notify-detailed'`

These settings have the same effect as the corresponding settings without `'-detailed'`, but give detailed exit messages instead of the default exit messages. For solvers without the new exit messages, the `'-detailed'` options give the same behavior as without `'-detailed'`.

### Messages in Output Structure

For solvers with enhanced exit messages, the `message` field of the output structure contains both the default (simpler) and the detailed exit messages, separated by a line of text stating `Stopping criteria details:`. The message field does not contain hyperlinks; it contains only text.

## Compatibility Considerations

For solvers with enhanced exit messages, the content of `output.message` contains many more characters than before. User code that relies on this field may need to be modified in order to display the larger exit message satisfactorily.

## Change in linprog Simplex Algorithm

The simplex algorithm of `linprog` now detects when there is no progress in the solution process. It attempts to continue by performing bound perturbation.

## Compatibility Considerations

The simplex algorithm of `linprog` might arrive at different solutions than before, and can solve more problems than before.

## Change in fminunc Exit Flag

One exit flag in the `fminunc` medium-scale solver was changed from –2 to 5. This flag appears when the solver predicts a change in function value at the next step in its iterations will be less than the `TolFun` tolerance. This condition can occur at a relative minimum, which should be reported by a positive flag.

## Compatibility Considerations

This change might cause users (or code) that examine exit flags to evaluate a result more favorably than previously, since positive exit flags represent normal termination of solvers.

## New demos

There are two new demos:

- A demo showing how to use Symbolic Math Toolbox™ functions to help calculate gradients and Hessians. Run the demo at the MATLAB command line by entering `echodemo symbolic_optim_demo`.
- A demo showing how to use `fseminf` for investigating the effect of parameter uncertainty. Run the demo at the MATLAB command line by entering `echodemo airpollution`.

Furthermore, the optimization tutorial demo now shows how to include extra parameters. Run the demo at the MATLAB command line by entering `echodemo tutdemo`.

# R2008b

## fsolve, lsqcurvefit, lsqnonlin Algorithm and Options Changes

- The Levenberg-Marquardt algorithm was refactored in the solvers `fsolve`, `lsqcurvefit` and `lsqnonlin`. It is now a more standard implementation, that accepts and preserves sparse Jacobians.
- Choose between the algorithms used in `fsolve`, `lsqcurvefit` and `lsqnonlin` using the new `Algorithm` option.
- There is a new `ScaleProblem` option that can sometimes help the Levenberg-Marquardt algorithm converge.
- The default `fsolve` algorithm, `'trust-region-dogleg'`, has been validated to work with sparse Jacobians.

## Compatibility Considerations

- The refactored Levenberg-Marquardt algorithm can cause `fsolve`, `lsqcurvefit` and `lsqnonlin` to yield different answers than before.
- The previous way of choosing the algorithm at the command line was to set the `LargeScale` option to `'on'` or `'off'`, and, for all solvers but `fsolve`, to set the `LevenbergMarquardt` option to `'on'` or `'off'`. For `fsolve`, in addition to the `LargeScale` option, you needed to set the `NonlEqnAlgorithm` option appropriately. `LargeScale`, `NonlEqnAlgorithm`, and `LevenbergMarquardt` are now ignored, except when choosing to use the Gauss-Newton algorithm.
- The Gauss-Newton algorithm warns that soon it may no longer be available.
- The default value of the `MaxFunEvals` option in the refactored Levenberg-Marquardt algorithm is now 200*`numberOfVariables`; the previous value was 100*`numberOfVariables`.

## Optimization Tool Enables Parallel Functionality

You can now access built-in parallel functionality in Optimization Tool for relevant Optimization Toolbox solvers and, if licensed, Global Optimization Toolbox solvers. The option is available when you have a license for Parallel Computing Toolbox™ functions.

## Central Finite Differences Available in Selected Solvers

The following solvers can now use central finite differences for gradient estimation:

- `fgoalattain`
- `fmincon`
- `fminimax`
- `fminunc`
- `fseminf`

The `fmincon` active-set algorithm and `fminunc` medium-scale algorithm gained central finite differences this release. The `fmincon` interior-point algorithm already had them, and the trust-region-reflective algorithm for both solvers requires a user-supplied gradient, so does not use finite differences.

To use central finite differences, use `optimset` to set the `FinDiffType` option to `'central'` instead of the default `'forward'`. This causes the solver to estimate gradients by formulae such as

$$\nabla f(x) \approx \left[\frac{f(x+\Delta_1 e_1) - f(x-\Delta_1 e_1)}{2\Delta_1}, \ldots, \frac{f(x+\Delta_n e_n) - f(x-\Delta_n e_n)}{2\Delta_n}\right],$$

instead of

$$\nabla f(x) \approx \left[\frac{f(x+\Delta_1 e_1) - f(x)}{\Delta_1}, \frac{f(x+\Delta_2 e_2) - f(x)}{\Delta_2}, \ldots, \frac{f(x+\Delta_n e_n) - f(x)}{\Delta_n}\right].$$

Central finite differences take twice as many function evaluations as forward finite differences, but are usually much more accurate.

Central finite differences can work in parallel for gradient estimation in `fgoalattain`, `fmincon` active-set algorithm, and `fminimax`. For details on how to use this parallel gradient estimation, see the Parallel Computing for Optimization chapter in the User's Guide.

## lsqnonneg Refactored

`lsqnonneg` was refactored. It can now use sparse matrices, and it preserves sparsity during its execution.

## Finite Difference Algorithm Tweaked

A subroutine for gradient estimation by forward finite differences in nonlinear solvers had a bug that affected it when the current point `x` had a component with the value 0. Forward finite differences are typically calculated with a step size proportional to `sqrt(eps)`, which is about $1.5*10^{-8}$. When a component of `x` was 0, the step size would instead be proportional to `DiffMinChange`, which has a default value of $10^{-8}$. There is now no difference in step size when `x` is 0.

## Compatibility Considerations

Nonlinear solvers can run slightly differently whenever an iteration causes a component of `x` to be zero, and gradients are estimated by forward finite differences.

## DerivativeCheck Tolerance Changed

The `DerivativeCheck` option enables you to ascertain whether the derivative (gradient) functions that you supply for objective or constraint functions give approximately the same values as those estimated by a solver using finite differences. The meaning of “approximately” has changed. Now it means the relative error of each component of the gradient is less than $10^{-6}$, unless the size of an analytically given component is smaller than 1, in which case it means the absolute difference is less than $10^{-6}$. Previously, the gradients were considered approximately equal if the maximum absolute error in any component of the gradient was less than ($10^{-6}$ * norm of analytic gradient) + $10^{-5}$.

## Compatibility Considerations

Some problems will now report violations of the `DerivativeCheck` condition, when previously they would not.

# R2008a

## Parallel Computing Toolbox Support in fmincon, fminimax, and fgoalattain

`fmincon`, `fminimax`, and `fgoalattain` can take finite differences in parallel in order to speed the estimation of gradients. For details on how to use this parallel gradient estimation, see the Parallel Computing for Optimization chapter in the User's Guide.

## Combined and Extended optimtool

The Global Optimization Toolbox GUIs `gatool` and `psearchtool` have been combined into the Optimization Tool GUI. To access these GUIs, type `optimtool` at the command line, and choose the appropriate solver.

Furthermore, three new Global Optimization Toolbox solvers were added to Optimization Tool: `gamultiobj`, `simulannealbnd`, and `threshacceptbnd`.

Optimization Tool shows Global Optimization Toolbox solvers only if these solvers are licensed.

## New fmincon Solver, New Option Algorithm for fmincon, Option LargeScale Changed

The new interior-point algorithm is a large-scale algorithm that can handle all types of constraints. It has several new options, explained in the `fmincon` function reference pages.

`fmincon` now has three algorithms. Choose between them by setting the new option `Algorithm` to:

- `'trust-region-reflective'` (formerly known as `'large scale'`)
- `'active-set'` (formerly known as `'medium scale'`)
- `'interior-point'`

By default, `Algorithm = 'trust-region-reflective'`.

## Compatibility Considerations

The previous way of choosing the algorithm at the command line was to set option `LargeScale` to `'on'` or `'off'`. `LargeScale` is now ignored, except when `LargeScale = 'off'` and `Algorithm = 'trust-region-reflective'`. In this case, the `'active-set'` algorithm is used, to minimize backward incompatibility.

## External Interface to KNITRO Libraries

Use the new `ktrlink` function to call KNITRO optimization libraries from Ziena Optimization, Inc. KNITRO libraries must be purchased separately. The External Interface chapter of the User's Guide describes the `ktrlink` function.

## Default PrecondBandWidth = Inf in lsqcurvefit, lsqnonlin, and fsolve

The default value of the `PrecondBandWidth` option changed from 0 to `Inf` for the `lsqcurvefit`, `lsqnonlin`, and `fsolve` solvers. This change was beneficial in the vast majority of tested problems.

In Optimization Tool, the default in **Algorithm settings > Subproblem algorithm** is now **Cholesky factorization**, instead of **Preconditioned CG** = 0.

## Compatibility Considerations

The new default can lead to slower performance for problems with high-dimensional nonlinearities. If this happens, change the default to another value such as 0 (the previous default).

## New Option TolConSQP with Incompatible Default Value

The new `TolConSQP` option exposes a parameter that was fixed at `eps` before. The parameter is used in the `fmincon`, `fminimax`, `fgoalattain`, and `fseminf` solvers.

## Compatibility Considerations

The new default value is `TolConSQP` = 1e–6. This did not affect a vast majority of tested cases, and was beneficial in some. If you want exactly the same behavior as before, set `TolConSQP = eps` using `optimset`.

## Field constrviolation in Output Structure

The `constrviolation` field now exists in the output structure for the `fgoalattain`, `fmincon`, `fminimax`, and `fseminf` functions; it measures the nonlinear constraint violation.

# R2007b

**Version: 3.1.2**

**Bug Fixes**

# R2007a

## Changes to Outputs of Multiobjective Solvers

- `fminimax` now returns the value of `max(fval)` in the output `maxfval`.
- The iterative display of `fminimax` and `fgoalattain` have changed.

## Compatibility Considerations

- The third output argument of the solver `fminimax`, `maxfval`, is described in the documentation as the maximum of the objective functions in the input `fun` evaluated at the solution `x`, that is, `max(fval)`. Before this release, `fminimax` actually returned the maximum of the objective functions in the reformulated minimax problem internally constructed by the algorithm. This value was typically very close to, but not necessarily equal to, `max(fval)`. `fminimax` now returns the exact value of `max(fval)` in the output `maxfval`.
- The iterative display for `fminimax` includes a new column with header `Objective value` that reports the objective function value of the nonlinear programming reformulation of the minimax problem. The column header `Max{F,constraints}` has been changed to `Max constraint`, and the column now contains the maximum violation among all constraints, both internally constructed and user-provided.

  The iterative display for `fgoalattain` now shows the value of the attainment factor in the `Attainment factor` column. A new column, `Max constraint`, contains the maximum violation among all constraints, both internally constructed and user-provided.

# R2006b

## New Optimization Tool

The Optimization Tool is a graphical user interface (GUI) for performing common optimization tasks with the Optimization Toolbox. Using the `optimtool`, you can do the following:

- Select a solver and define your optimization problem.
- Set and inspect optimization options and their default values.
- Run problems and visualize results.
- Import and export problem definitions, algorithm options, and results between the MATLAB workspace and the Optimization Tool.
- Automatically generate M-code to capture, automate, and recreate your problem.
- Access built-in help.

## Plot Functions Option Added

You can now specify the `PlotFcns` option in the `optimset` function or using the Optimization Tool for use with an Optimization Toolbox solver. With this option, you can plot various measures of progress while the algorithm executes. You can select from several predefined plots, or you can write your own.

## Output Function Option Enhanced to Accept Multiple Functions

You can now specify more than one output function in the `OutputFcn` option.

## Changes to the Output Function

The output function input `x` and fields in the `optimValues` structure have the following changes that address bugs in previous releases:

- `residual` now returns the residual vector for `lsqnonlin` and `lsqcurvefit`.
- `resnorm` contains the sum of squares and has been added for `lsqnonlin` and `lsqcurvefit`. The previous field `fval` has been removed for these functions.
- `procedure` has been removed for `lsqnonlin`, `lsqcurvefit`, and `fsolve`.
- `x` now returns the expected shape and size for `fgoalattain` and `fminimax`.

## Compatibility Considerations

The above changes to the input `x` and `optimValues` structure have the following compatibility considerations in the output function:

- If you have references to the `residual` in a previous version, note that the value of this field has changed for `lsqnonlin` and `lsqcurvefit`. This fixes the problem addressed by the bug report S-289285.
- Any references to `fval` for `lsqnonlin` and `lsqcurvefit` need to be updated to `resnorm`. This fixes the problem addressed by the bug report S-289285.
- Any references to `procedure` for `lsqnonlin` and `lsqcurvefit` need to be removed. This fixes the problem addressed by the bug report S-291974.

- Previously, for `fgoalattain` and `fminimax`, `x` returned a column vector with an additional last element. If you have references to the values for `x` in a previous version, the extra element must be removed and the output vector may need to be reshaped. This fixes the problem addressed by the bug report S-315658.

# R2006a

**Version: 3.0.4**

**Bug Fixes**

# R14SP3

**Version: 3.0.3**

**New Features**

**Bug Fixes**

## Notify Parameter Added to Display Option for Five Functions

You can now set the optimization option `Display` to `'notify'` for the functions `fmincon`, `fminunc`, `fminimax`, `fgoalattain`, and `fseminf`. When `Display` is set to `'notify'`, the output is displayed only if the function does not converge.