

Symbolic Math Toolbox™ Release Notes



MATLAB®

How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Symbolic Math Toolbox™ Release Notes

© COPYRIGHT 2004–2017 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

R2017b

Unit Systems: Convert between SI and US units and create custom systems of units	1-2
Unit Information: Get information on units and physical dimensions with the unitInfo function	1-2
Symbolic String Evaluation: Evaluate strings as symbolic expressions with the str2sym function	1-2
Special Functions: Calculate the Meijer G-function, elliptic nome function, Jacobi zeta function, and Jacobi elliptic functions	1-2
Code Generation Comments: Add comments to code generated from symbolic expressions	1-3
Functionality being removed or changed	1-3

R2017a

Units: Use physical units in symbolic calculations with the symunit function	2-2
Live Scripts: Convert more MuPAD notebooks to MATLAB live scripts with the convertMuPADNotebook function, including notebooks with MuPAD procedures	2-2

Isolate Variables: Rearrange equation to isolate a variable or expression on the left side	2-2
Decompose Equations: Extract the left and right side of an equation with the lhs and rhs functions	2-2
Fibonacci Numbers: Compute Fibonacci numbers with fibonacci	2-3
Functionality being removed or changed	2-3

R2016b

MATLAB Live Scripts: Convert more MuPAD notebooks automatically to MATLAB live scripts using the convertMuPADNotebook function	3-2
Piecewise Expressions: Define conditional symbolic expressions with the piecewise function	3-2
Plotting Implicit Functions: Plot implicit symbolic functions in 2-D and 3-D with MATLAB fimplicit and fimplicit3 functions	3-2
Numerical Integration: Integrate symbolic expressions using variable-precision arithmetic with the vpaintegral function	3-2
Prime Numbers: Find prime numbers with MATLAB prevprime and nextprime functions	3-2
Fold vector: Combine elements of vector with MATLAB fold function	3-2
Simscape Component Variables in MATLAB Workspace: Load the names of the component variables as symbolic functions	3-3
Functionality being removed or changed	3-3

Live Scripts: Edit symbolic code and visualize results in MATLAB Live Editor, and convert MuPAD notebooks to MATLAB live scripts	4-2
Plotting: Create 2-D, 3-D, contour, surface, and mesh plots with MATLAB fplot, fplot3, fcontour, fsurf, and fmesh functions	4-2
Simscape Component Generation: Create custom components directly from symbolic math equations for use in dynamic simulation	4-2
MATLAB cell2sym and sym2cell simplify conversions between symbolic and cell arrays	4-3
MATLAB nchoosek accepts a vector as its first argument . . .	4-3
MATLAB sym creates multidimensional arrays	4-3
Functionality being removed or changed	4-4

Fourier and Laplace transforms and their inverses for a wider variety of input expressions, including hyperbolic functions	5-2
MATLAB series function for computing Puiseux series expansion	5-2
MATLAB hermiteForm and smithForm functions for computing Hermite and Smith normal forms of matrices	5-2

Sparse argument for matlabFunction, odeFunction, and daeFunction for using sparse instead of dense matrices in generated MATLAB functions	5-2
MATLAB has function for searching subexpressions in a symbolic expression	5-3
MATLAB root function for representing roots of polynomials	5-3
New Symbolic Math Toolbox examples	5-3
Functionality being removed or changed	5-4

R2015a

MATLAB functionalDerivative function for finding derivatives of functionals	6-2
MATLAB odeFunction for converting systems of algebraic expressions to MATLAB functions suitable for ode45 and other ODE solvers	6-2
MATLAB partfrac function for computing partial fraction decomposition	6-2
MATLAB sympref function for specifying preferences for symbolic functions fourier, ifourier, and heaviside	6-2
Optimize argument for controlling code optimization in generated MATLAB functions returned by matlabFunction, odeFunction, and daeFunction	6-2
MuPAD isolate function for rearranging an equation so that the variable or expression appears on the left side	6-3
FactorMode argument offering different modes of factorization returned by MATLAB factor function	6-3

Reverse accumulation option for cumsum and cumprod functions	6-3
MATLAB functions chol, lu, qr, and rank now return certain outputs as type double	6-3
MATLAB functions assume, assumeAlso, assumptions, sym, and syms have changes to assumptions mechanism	6-4
MATLAB function combine combines additional expressions	6-4
New and updated Symbolic Math Toolbox examples	6-4
MATLAB solve function uses default MaxDegree value of 2	6-5
MuPAD functions taylor and mtaylor error when they cannot find a Taylor series	6-6
MuPAD orthogonal polynomial functions return polynomial expressions	6-6
MATLAB function sym treats i in character vectors as a variable	6-6
Functionality being removed or changed	6-7

R2014b

MATLAB solve function returning parameters and conditions in solutions	7-2
Functions for analyzing and reducing systems of differential algebraic equations (DAEs), such as isLowIndexDAE and reduceDAEIndex	7-2

MATLAB functions representing orthogonal polynomials: chebyshevT, chebyshevU, legendreP, laguerreL, hermiteH, jacobiP, and gegenbauerC	7-3
MATLAB pade function for computing Padé approximation	7-4
funm function for computing matrix functions	7-4
MATLAB kummerU function for computing confluent hypergeometric (Kummer U) function	7-4
MATLAB polylog function for computing polylogarithms ...	7-4
MATLAB signIm function for computing signs of imaginary parts of complex numbers	7-4
MATLAB in function for representing conditions on symbolic inputs	7-4
MATLAB divisors function for finding divisors of integers and polynomials	7-5
MATLAB functions nnz and nonzeros for finding nonzero elements in a symbolic array	7-5
MATLAB pochhammer function to calculate Pochhammer symbol	7-5
MATLAB kroneckerDelta function for computing the Kronecker delta function	7-5
MATLAB dirac function with two input arguments for computing derivatives of the Dirac delta function	7-5
MATLAB isAlways function warns when returning false for undecidable inputs	7-6
MuPAD generate::fortran function can use Fortran 90 as the target compiler	7-6
MATLAB mod function for computing modulus after division	7-6

MATLAB <code>gcd</code> and <code>lcm</code> functions accept vectors and matrices	7-7
MATLAB <code>rem</code> function accepts vectors and matrices	7-8
MATLAB <code>factor</code> function only accepts scalar inputs and returns vector of factors of input	7-8
MuPAD Notebook app supports left and right double square brackets	7-8
Functionality being removed or changed	7-9

R2014a

MATLAB functions for computing special integrals, gamma functions, dilogarithm function, and number-theoretic functions	8-2
MATLAB function <code>qr</code> for computing symbolic QR factorization	8-2
MATLAB function <code>combine</code> for combining symbolic expressions with multiple calls to the same function	8-3
MATLAB functions <code>max</code> and <code>min</code> for finding the largest and smallest elements of a symbolic array	8-3
<code>vpasolve</code> can use random starting points when searching for solutions	8-3
Support for Unicode characters in MuPAD that includes using Asian language characters in character vectors and text	8-3
Support for specifying encoding in MuPAD file operations	8-4

Choice of right- or left-handed spherical coordinate system for the MuPAD vector analysis functions	8-4
MATLAB special functions and functions for computing integral and Z-transforms accept several nonscalar arguments	8-5
MATLAB function erfc accepts two arguments	8-5
Functionality being removed or changed	8-6

R2013b

MATLAB evaluateMuPADNotebook and allMuPADNotebooks functions to evaluate MuPAD notebooks and return list of open notebooks	9-2
bernstein function for approximating functions using Bernstein polynomials, and bernsteinMatrix function for computing Bezier curves	9-2
MATLAB cumsum and cumprod functions for computing cumulative sums and products	9-2
MATLAB isfinite, isinf, and isnan functions for testing for finite, infinite, and NaN elements in symbolic arrays	9-2
ExclusiveConditions option that makes MuPAD piecewise function equivalent to an if-elif-end_if statement	9-2
MATLAB mupadNotebookTitle function to find the window title of the MuPAD notebook	9-3
MATLAB close function to close MuPAD notebooks from MATLAB	9-3
diff supports mixed derivatives	9-3

coeffs function extracts coefficients of multivariate polynomials	9-3
linspace, logspace, and compan functions for symbolic objects	9-4
Indexing uses lists, vectors, and matrices of indices	9-4
MuPAD lets you set assumptions on matrices	9-4
int, symprod, and symsum let you specify lower and upper bounds as vectors	9-4
Functionality being removed or changed	9-4

R2013a

Linear algebra functions for computing matrix factorizations (lu, chol), pseudoinverse, orthogonal basis, and adjoint	10-2
Verification of solutions of systems of equations and arbitrary symbolic function substitution in subs function	10-2
Simplification for more types of trigonometric and hyperbolic expressions and expressions with nested roots	10-2
Special functions for computing polar angle, atan2 function, imaginary error function, and exponential and elliptic integrals	10-3
toeplitz function for creating Toeplitz matrices	10-4
sqrtm function for computing square roots of matrices	10-4
sign function for computing signs of numbers	10-4

Real option of the <code>linalg::orthog</code> function for avoiding complex conjugates	10-4
Real option of the <code>linalg::factorCholesky</code> function for avoiding complex conjugates	10-5
New arguments of the <code>svd</code> function for computing the “economy size” singular value decomposition	10-5
<code>isequaln</code> function for testing equality of symbolic objects	10-5
Control over the order in which <code>solve</code> and <code>vpasolve</code> functions return solutions	10-5
Functionality being removed or changed	10-6

R2012b

MATLAB symbolic matrix analysis functions for characteristic (<code>charpoly</code>) and minimal (<code>minpoly</code>) polynomials and for norm (<code>norm</code>) and condition (<code>cond</code>) number	11-2
<code>poles</code> function for determining the poles of an expression	11-2
<code>vpasolve</code> function for solving equations and systems using variable precision arithmetic	11-2
Functions for converting linear systems of equations to matrix form $AX=B$ (<code>equationsToMatrix</code>) and solving matrix equations (<code>linsolve</code>)	11-2
MATLAB symbolic functions for describing pulses: <code>rectangularPulse</code> and <code>triangularPulse</code>	11-3
MuPAD functions for computing integral and Z-transforms	11-3

MuPAD Pref::fourierParameters function for specifying Fourier parameters	11-3
MuPAD functions for adding transform patterns	11-3
noFlatten option of the MuPAD proc function for preventing sequence flattening	11-4
testtype uses testtypeDom slot for overloading by the second argument	11-4
New upper limit on the number of digits in double	11-4
New definition for real and imag	11-5
Functionality being removed or changed	11-6

R2012a

New Special Functions	12-2
New Vector Analysis Functions	12-2
Computations with Symbolic Functions	12-2
Assumptions on Variables	12-3
New Relational Operators Create Equations, Inequalities, and Relations	12-3
New Logical Operators Create Logical Expressions	12-3
New Functions Test Validity of Symbolic Equations, Inequalities, and Relations	12-4
New Functions Manipulate Symbolic Expressions	12-4

New odeToVectorField Function Converts Higher-Order Differential Equations to Systems of First-Order Differential Equations	12-4
New Calling Syntax for the taylor Function	12-5
New MuPAD Functions Compute Rectangular and Triangular Pulse Functions	12-5
MuPAD det, linalg::det, inverse, linsolve, and linalg::matlinsolve Functions Accept the New Normal Option	12-5
MuPAD linalg::matlinsolve Function Accepts the New ShowAssumptions Option	12-6
Enhanced MuPAD pdivide Function	12-6
Improved MuPAD prog::remember Function	12-6
Functionality Being Removed or Changed	12-6

R2011b

MATLAB Editor Now Supports MuPAD Program Files	13-2
dsolve, expand, int, simple, simplify, and solve Accept More Options	13-2
New read Function Reads MuPAD Program Files in MATLAB	13-2
New symprod Function Computes Products of Series	13-2
New hessian Function Computes Hessian Matrices	13-2
New gradient Function Computes Vector Gradients	13-3

New erfc Function Computes the Complementary Error Function	13-3
New psi Function Computes the Digamma and Polygamma Functions	13-3
New wrightOmega Function Computes the Wright omega Function	13-3
New simplifyFraction Function Simplifies Expressions ...	13-3
New MuPAD simplifyRadical Function Simplifies Radicals in Arithmetical Expressions	13-3
pretty Function Now Uses Abbreviations in Long Output Expressions for Better Readability	13-3
MuPAD normal Function Accepts the New Expand Option	13-4
Modified MuPAD groebner Library	13-4
MuPAD groebner::gbasis Function Accepts the New Factor and IgnoreSpecialCases Options	13-4
New MuPAD Functions for Computing Logarithms	13-5
Functionality Being Removed or Changed	13-5

R2011a

Expression Wrapping of Math Output in the MuPAD Notebook Interface	14-2
Symbolic Solver Handles More Non-Algebraic Equations ..	14-2
Improved Performance in the Ordinary Differential Equation Solver	14-2

Improved Performance for Polynomial Arithmetic Operations	14-2
New MuPAD polylib::subresultant Function Computes Subresultants of Polynomials	14-2
MuPAD partfrac Function Accepts the New List Option ...	14-2
New MuPAD inverf and inverfc Functions Compute the Inverses of Error Functions	14-2
New MuPAD numlib::checkPrimalityCertificate Function Tests Primality Certificates	14-3
New Demos	14-3
Functionality Being Removed or Changed	14-3

R2010b

sym Function Creates Matrices of Symbolic Variables	15-2
generate::Simscape Function Generates Simscape Equations from MuPAD Expressions	15-2
MuPAD Code Generation Functions Accept the New NoWarning Option	15-2
Improved MuPAD Hyperlink Dialog Box	15-2
MuPAD Notebook Highlights Matched and Unmatched Delimiters	15-2
Improved Performance When Solving Linear Systems in a Matrix Form	15-2
MuPAD Solver for Ordinary Differential Equations Handles More Equation Types	15-3

New Syntax for the MuPAD prog::getOptions Function	15-3
New Syntax for the MuPAD prog::trace Function	15-3
Improved Interface for Arithmetical Operations on Polynomials	15-3
MuPAD igcd Function Now Accepts Complex Numbers as Arguments	15-4
Enhanced Solver For Factorable Polynomial Systems	15-4
MuPAD Now Evaluates Large Sums with Subtractions Faster	15-4
MuPAD freeIndets Function Accepts the New All Option . .	15-4
Functionality Being Removed or Changed	15-4

R2010a

When Opening Notebook, MuPAD Can Jump to Particular Locations	16-2
simscapeEquation Function Generates Simscape Equations from Symbolic Expressions	16-2
New Calling Syntax for the sort Function	16-2
Changes in the symengine Function	16-2
64-Bit GUI Support for Macintosh	16-2
New MuPAD Print Preview Dialog	16-2
Improved Configure MuPAD Dialog Box	16-3

MuPAD Support for Basic Arithmetic Operations for Lists	16-3
Improved Performance When Operating on Matrices with Symbolic Elements	16-3
Enhanced MuPAD divide Function	16-3
Improved Performance for Operations on Polynomials	16-3
MuPAD coeff Function Accepts the New All Option	16-4
MuPAD expand Function Accepts the New ArithmeticOnly Option	16-4
MuPAD expand Function Now Expands Powers of Products	16-4
New Calling Syntax for MuPAD rationalize Function	16-4
Enhanced MuPAD simplify and Simplify Functions	16-4
MuPAD subs Function Accepts the New EvalChanges Option	16-5
MuPAD Solver for Ordinary Differential Equations Handles More Equation Types	16-5
Functionality Being Removed or Changed	16-5

R2009b

Support for Windows x64 and 64-Bit Macintosh	17-2
sym and syms Use Reserved Words as Variable Names	17-2
Toolbox Now Displays Floating-Point Results with Their Original Precision	17-2

New MuPAD Preference Pref::outputDigits Controls Floating-Point Outputs	17-3
Solver for Ordinary Differential Equations Handles More Equation Types	17-3
MuPAD limit Function Supports Limits for Incomplete Gamma Function and Exponential Integral Function ...	17-3
Enhanced Simplification Routines for MuPAD Special Functions	17-3
Enhanced MuPAD combine Function for Logarithms	17-3
MuPAD normal Function Accepts New Options	17-3
Functionality Being Removed or Changed	17-4

R2009a

dsolve Accepts the New Option IgnoreAnalyticConstraints	18-2
emlBlock Function Generates Embedded MATLAB Function Blocks from Symbolic Objects	18-2
matlabFunction Improves Control over Input and Output Parameters	18-2
Enhancements to Object-Oriented Programming Capabilities	18-3
generate::MATLAB Function Converts MuPAD Expressions to MATLAB Code	18-3
MuPAD IgnoreAnalyticConstraints Option Specifies That Core Functions Apply Common Algebraic Assumptions to Simplify Results	18-3

MuPAD Outputs Contain Abbreviations for Better Readability	18-3
MuPAD Solver for Ordinary Differential Equations Handles More Equation Types	18-4
MuPAD limit Function Now Can Compute Limits for Piecewise Functions	18-4
New and Improved MuPAD Special Functions	18-4
New Calling Syntax for Test Report Function prog::tcov ..	18-4
New Demos	18-4

R2008b

Bug Fixes

R2008a+

Bug Fixes

R2007b+

MuPAD Engine Replaces Maple Engine	21-2
New MuPAD Language and Libraries Supplant Extended Symbolic Math Toolbox Software	21-5
New MuPAD Help Viewer (GUI)	21-6

New MuPAD Notebook Interface (GUI)	21-6
New MuPAD Editor and Debugger (GUI)	21-6
New Functionality for Communication Between MATLAB Workspace and MuPAD	21-7
New symengine Command for Choosing a Maple Engine ..	21-7
New matlabFunction Generates MATLAB Functions	21-7

R2008a

Bug Fixes

R2007b

Bug Fixes

R2007a

Maple10 Access Added for Linux 64-bit Processors and Intel Macintosh Platforms	24-2
---	-------------

**Change in call to code generation package using the maple
function** 25-2

R2017b

Version: 8.0

New Features

Bug Fixes

Compatibility Considerations

Unit Systems: Convert between SI and US units and create custom systems of units

Automatically convert units between the unit systems SI, CGS, and US. For details, see “Unit Conversions and Unit Systems”. Additionally, define custom unit systems by using `newUnitSystem` and then convert units to the custom unit systems.

- `baseUnits` returns the base units of a unit system.
- `derivedUnits` returns the derived units of a unit system.
- `newUnitSystem` creates a custom unit system.
- `rewrite` converts units between unit systems.
- `removeUnit` removes custom units created with `newUnit`.
- `removeUnitSystem` removes custom unit systems created with `newUnitSystem`.
- `unitSystems` lists available unit systems.

Unit Information: Get information on units and physical dimensions with the `unitInfo` function

`unitInfo` returns information on available units, dimensions that have available units, and the available units for a given dimension.

Symbolic String Evaluation: Evaluate strings as symbolic expressions with the `str2sym` function

Evaluate strings as symbolic expressions with `str2sym`.

Special Functions: Calculate the Meijer G-function, elliptic nome function, Jacobi zeta function, and Jacobi elliptic functions

These special functions are available:

- `ellipticNome` returns the elliptic nome function.
- `jacobiAM` returns the Jacobi amplitude function.
- `jacobiCD` returns the Jacobi CD function.

- `jacobiCN` returns the Jacobi CN function.
- `jacobiCS` returns the Jacobi CS function.
- `jacobiDC` returns the Jacobi DC function.
- `jacobiDN` returns the Jacobi DN function.
- `jacobiDS` returns the Jacobi DS function.
- `jacobiNC` returns the Jacobi NC function.
- `jacobiND` returns the Jacobi ND function.
- `jacobiNS` returns the Jacobi NS function.
- `jacobiSC` returns the Jacobi SC function.
- `jacobiSD` returns the Jacobi SD function.
- `jacobiSN` returns the Jacobi SN function.
- `jacobiZeta` returns the Jacobi zeta function.
- `meijerG` returns the Meijer G-function.

Code Generation Comments: Add comments to code generated from symbolic expressions

You can add comments to code generated with code generation functions by using the 'Comments' option. The code generation functions are: `ccode`, `daefunction`, `fortran`, `matlabFunction`, `matlabFunctionBlock`, and `odefunction`.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>eval</code> for symbolic inputs	Still runs	Use <code>subs</code> to replace symbolic variables by their values. Use <code>double</code> to convert symbolic objects to double.	

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD® notebooks	Still runs	MATLAB® live scripts. For details, see “Convert MuPAD Notebooks to MATLAB Live Scripts”.	MATLAB live scripts support most MuPAD functionality, though there are some differences.
Character vectors as inputs to <code>dsolve</code>	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>dsolve('Dy = y')</code> with <code>syms y(t); dsolve(diff(y,t) == y)</code> .	Do not specify equations and variables as character vectors. Instead, create symbolic variables using <code>syms</code> , and then pass them as comma-separated input arguments, or as a vector of input arguments.
Character vectors as inputs to <code>odeToVectorField</code>	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>odeToVectorField('D2y = x')</code> with <code>syms y(x); odeToVectorField(diff(y,x,2) == x)</code> .	Do not specify equations and variables as character vectors. Instead, create symbolic variables using <code>syms</code> , and then pass them as comma-separated input arguments, or as a vector of input arguments.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<p>Character vector inputs to <code>sym</code> and <code>vpa</code> are restricted to numbers and valid variable names.</p>	<p>Warns</p>	<p>When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:</p> <ul style="list-style-type: none"> • <code>syms x; x + 1</code> <p>instead of</p> <pre>sym('x + 1')</pre> <ul style="list-style-type: none"> • <code>exp(sym(pi))</code> <p>instead of</p> <pre>sym('exp(pi)')</pre> <ul style="list-style-type: none"> • <code>syms f(var1,...varN)</code> <p>instead of</p> <pre>f(var1,...varN) = sym('f(var1,...varN)')</pre> <ul style="list-style-type: none"> • <code>vpa((1 + sqrt(sym(5)))/2)</code> <p>instead of</p> <pre>vpa('(1 + sqrt(5))/2')</pre>	<p>Support of character vectors that are not valid variable names and do not define a number will be removed in a future release. To create symbolic expressions, first create symbolic variables, and then use operations on them.</p>
<p><code>ezplot</code></p>	<p>Still runs</p>	<p><code>fplot</code>, <code>fimplicit</code></p>	<p>Replace <code>ezplot</code> by <code>fplot</code> for 2-D line plots and by <code>fimplicit</code> for 2-D implicit plots.</p>

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
ezplot3	Still runs	fplot3	Replace ezplot3 by fplot3.
ezsurf and ezsurfz	Still runs	fsurf	Replace ezsurf by fsurf. Replace ezsurfz by fsurf(..., 'ShowContours', 'on').
ezcontour and ezcontourf	Still runs	fcontour	Replace ezcontour with fcontour. Replace ezcontourf by fcontour(..., 'Fill', 'on').
ezmesh and ezmeshz	Still runs	fmesh	Replace ezmesh by fmesh. Replace ezmeshz by fmesh(..., 'ShowContours', 'on').
Character vectors as inputs to solve	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace solve('2*r = 1', 'r') with syms r; solve(2*r == 1, r).	Do not specify equations and variables as character vectors. Instead, create symbolic variables using syms, and then pass them as comma-separated input arguments, or as a vector of input arguments.
findsym	Warns	symvar	Replace all instances of findsym with symvar.
mfun	Warns	Appropriate special function. For example, replace mfun('P', 1, 2, 3, 4) with jacobiP(1, 2, 3, 4).	Replace all instances of mfun with the appropriate function call. See “Mathematical Functions” for the list of available special functions.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
mfunlist	Warns	See “Mathematical Functions”.	See “Mathematical Functions” for the list of available special functions.

R2017a

Version: 7.2

New Features

Bug Fixes

Compatibility Considerations

Units: Use physical units in symbolic calculations with the `symunit` function

The `symunit` function specifies physical units symbolically. For details, see the Units of Measurement Tutorial. The following functions manipulate these symbolic units:

- `checkUnits` checks units for compatible and consistent dimensions.
- `findUnits` finds units in input.
- `isUnit` determines if the input is a symbolic unit.
- `newUnit` defines new units.
- `separateUnits` separates units from symbolic expressions.
- `str2symunit` converts character vectors to units.
- `symunit2str` converts units to character vectors.
- `unitConversionFactor` returns the conversion factor between compatible units.

Live Scripts: Convert more MuPAD notebooks to MATLAB live scripts with the `convertMuPADNotebook` function, including notebooks with MuPAD procedures

Use `convertMuPADNotebook` to convert MuPAD notebooks to MATLAB live scripts. For more information, see [Convert MuPAD Notebooks to MATLAB Live Scripts](#).

Isolate Variables: Rearrange equation to isolate a variable or expression on the left side

The `isolate` function rearranges an equation so that the variable or expression appears on the left side of the equation. If `isolate` cannot isolate the variable or expression, it moves all terms containing the variable or expression to the left side.

Decompose Equations: Extract the left and right side of an equation with the `lhs` and `rhs` functions

Use `lhs` and `rhs` to extract the left and right sides of equations.

Fibonacci Numbers: Compute Fibonacci numbers with fibonacci

The `fibonacci` function computes the Fibonacci numbers.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD notebooks	Still runs	MATLAB live scripts. For details, see Convert MuPAD Notebooks to MATLAB Live Scripts.	MATLAB live scripts support most MuPAD functionality, though there are some differences.
Character vectors as inputs to <code>dsolve</code>	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>dsolve('Dy = y')</code> with <code>syms y(t); dsolve(diff(y,t) == y)</code> .	Do not specify equations and variables as character vectors. Instead, create symbolic variables using <code>syms</code> , and then pass them as comma-separated input arguments, or as a vector of input arguments.
Character vectors as inputs to <code>odeToVectorField</code>	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>odeToVectorField('D2y = x')</code> with <code>syms y(x); odeToVectorField(diff(y,x,2) == x)</code> .	Do not specify equations and variables as character vectors. Instead, create symbolic variables using <code>syms</code> , and then pass them as comma-separated input arguments, or as a vector of input arguments.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<p>Character vector inputs to <code>sym</code> and <code>vpa</code> are restricted to numbers and valid variable names.</p>	<p>Warns</p>	<p>When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:</p> <ul style="list-style-type: none"> • <code>syms x; x + 1</code> <p>instead of</p> <pre>sym('x + 1')</pre> <ul style="list-style-type: none"> • <code>exp(sym(pi))</code> <p>instead of</p> <pre>sym('exp(pi)')</pre> <ul style="list-style-type: none"> • <code>syms f(var1,...varN)</code> <p>instead of</p> <pre>f(var1,...varN) = sym('f(var1,...varN)')</pre> <ul style="list-style-type: none"> • <code>vpa((1 + sqrt(sym(5)))/2)</code> <p>instead of</p> <pre>vpa('(1 + sqrt(5))/2')</pre>	<p>Support of character vectors that are not valid variable names and do not define a number will be removed in a future release. To create symbolic expressions, first create symbolic variables, and then use operations on them.</p>
<p><code>ezplot</code></p>	<p>Still runs</p>	<p><code>fplot</code>, <code>fimplicit</code></p>	<p>Replace <code>ezplot</code> by <code>fplot</code> for 2-D line plots and by <code>fimplicit</code> for 2-D implicit plots.</p>

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
ezplot3	Still runs	fplot3	Replace ezplot3 by fplot3.
ezsurf and ezsurfz	Still runs	fsurf	Replace ezsurf by fsurf. Replace ezsurfz by fsurf(..., 'ShowContours', 'on').
ezcontour and ezcontourf	Still runs	fcontour	Replace ezcontour with fcontour. Replace ezcontourf by fcontour(..., 'Fill', 'on').
ezmesh and ezmeshz	Still runs	fmesh	Replace ezmesh by fmesh. Replace ezmeshz by fmesh(..., 'ShowContours', 'on').
Character vectors as inputs to solve	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace solve('2*r = 1', 'r') with syms r; solve(2*r == 1, r).	Do not specify equations and variables as character vectors. Instead, create symbolic variables using syms, and then pass them as comma-separated input arguments, or as a vector of input arguments.
findsym	Warns	symvar	Replace all instances of findsym with symvar.
mfun	Warns	Appropriate special function. For example, replace mfun('P', 1, 2, 3, 4) with jacobiP(1, 2, 3, 4).	Replace all instances of mfun with the appropriate function call. See Mathematical Functions for the list of available special functions.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
mfunlist	Warns	See Mathematical Functions.	See Mathematical Functions for the list of available special functions.

R2016b

Version: 7.1

New Features

Bug Fixes

Compatibility Considerations

MATLAB Live Scripts: Convert more MuPAD notebooks automatically to MATLAB live scripts using the `convertMuPADNotebook` function

Use `convertMuPADNotebook` to convert MuPAD notebooks to MATLAB live scripts. For more information, see [Convert MuPAD Notebooks to MATLAB Live Scripts](#).

Piecewise Expressions: Define conditional symbolic expressions with the `piecewise` function

Use `piecewise` to define conditional symbolic expressions or functions, called piecewise expressions or functions.

Plotting Implicit Functions: Plot implicit symbolic functions in 2-D and 3-D with MATLAB `fimplicit` and `fimplicit3` functions

Use `fimplicit` and `fimplicit3` to plot implicit functions in 2-D and 3-D.

Numerical Integration: Integrate symbolic expressions using variable-precision arithmetic with the `vpaintegral` function

Use `vpaintegral` to perform high-precision numerical integration using variable-precision arithmetic.

Prime Numbers: Find prime numbers with MATLAB `prevprime` and `nextprime` functions

Use `nextprime` and `prevprime` to find the nearest prime numbers above and below a given number respectively.

Fold vector: Combine elements of vector with MATLAB `fold` function

Use `fold` to combine (fold) the elements of a vector by applying a given function to the elements pairwise from left to right.

Simscape Component Variables in MATLAB Workspace: Load the names of the component variables as symbolic functions

`symReadSSCVariables` accepts the name-value pair argument, `'ReturnFunctions', true`. When you use this argument, `symReadSSCVariables` returns the names of the variables of a Simscape™ component as a cell array of symbolic functions, such as $v(t)$, $f(t)$, and so on. The independent variable is always t . Without this argument, `symReadSSCVariables` returns the names of the variables as a cell array of symbolic variables, such as v , f , and so on.

To create individual symbolic variables or functions from the elements of resulting cell arrays in the MATLAB workspace, use `syms`. For example, if `symReadSSCVariables` returns the names of the variables as a cell array `names`, use `syms(names)`.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD notebooks	Still runs	MATLAB live scripts. For details, see Convert MuPAD Notebooks to MATLAB Live Scripts .	MATLAB live scripts support most MuPAD functionality, though there are some differences.
Character vectors as inputs to <code>dsolve</code>	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>dsolve('Dy = y')</code> with <code>syms y(t); dsolve(diff(y,t) == y)</code> .	Do not specify equations and variables as character vectors. Instead, create symbolic variables using <code>syms</code> , and then pass them as input arguments separated by commas, or as a vector of input arguments.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<p>Character vector inputs to <code>sym</code> and <code>vpa</code> are restricted to numbers and valid variable names.</p>	<p>Warns</p>	<p>When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:</p> <ul style="list-style-type: none"> • <code>syms x; x + 1</code> <p>instead of</p> <pre>sym('x + 1')</pre> <ul style="list-style-type: none"> • <code>exp(sym(pi))</code> <p>instead of</p> <pre>sym('exp(pi)')</pre> <ul style="list-style-type: none"> • <code>syms f(var1,...varN)</code> <p>instead of</p> <pre>f(var1,...varN) = sym('f(var1,...varN)')</pre> <ul style="list-style-type: none"> • <code>vpa((1 + sqrt(sym(5)))/2)</code> <p>instead of</p> <pre>vpa('(1 + sqrt(5))/2')</pre>	<p>Support of character vectors that are not valid variable names and do not define a number will be removed in a future release. To create symbolic expressions, first create symbolic variables, and then use operations on them.</p>
<p><code>ezplot</code></p>	<p>Still runs</p>	<p><code>fplot</code>, <code>fimplicit</code></p>	<p>Replace <code>ezplot</code> by <code>fplot</code> for 2-D line plots and by <code>fimplicit</code> for 2-D implicit plots.</p>

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
ezplot3	Still runs	fplot3	Replace ezplot3 by fplot3.
ezsurf and ezsurfz	Still runs	fsurf	Replace ezsurf by fsurf. Replace ezsurfz by fsurf(..., 'ShowContours', 'on').
ezcontour and ezcontourf	Still runs	fcontour	Replace ezcontour with fcontour. Replace ezcontourf by fcontour(..., 'Fill', 'on').
ezmesh and ezmeshz	Still runs	fmesh	Replace ezmesh by fmesh. Replace ezmeshz by fmesh(..., 'ShowContours', 'on').
MuPAD: transpose(1) and transpose(x)	Still runs	Not applicable.	MuPAD: transpose(1) and tranpose(x) now return 1 and x instead of the function call.
Character vectors as inputs to solve	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace solve('2*r = 1', 'r') with syms r; solve(2*r == 1, r).	Do not specify equations and variables as character vectors. Instead, create symbolic variables using syms, and then pass them as input arguments separated by commas, or as a vector of input arguments.
findsym	Warns	symvar	Replace all instances of findsym with symvar.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>mfun</code>	Warns	Appropriate special function. For example, replace <code>mfun('P', 1, 2, 3, 4)</code> with <code>jacobiP(1, 2, 3, 4)</code> .	Replace all instances of <code>mfun</code> with the appropriate function call. See Mathematical Functions for the list of available special functions.
<code>mfunlist</code>	Warns	See Mathematical Functions .	See Mathematical Functions for the list of available special functions.

R2016a

Version: 7.0

New Features

Bug Fixes

Compatibility Considerations

Live Scripts: Edit symbolic code and visualize results in MATLAB Live Editor, and convert MuPAD notebooks to MATLAB live scripts

The MATLAB `convertMuPADNotebook` function converts the code from MuPAD notebooks (file extension `.mn`) to MATLAB live script files (file extension `.mlx`). The function also flags the code lines and formatting that cannot be directly translated from the MuPAD language to the MATLAB language. For information on live scripts, see [Create Live Scripts](#).

Plotting: Create 2-D, 3-D, contour, surface, and mesh plots with MATLAB `fplot`, `fplot3`, `fcontour`, `fsurf`, and `fmesh` functions

New MATLAB functions to plot mathematical expressions. These functions supersede the existing `ez` family of plotting functions, such as `ezplot`. The `ez` functions remain available.

- `fplot` plots 2-D lines, including parametric lines. Supersedes `ezplot`.
- `fplot3` plots 3-D parametric curves. Supersedes `ezplot3`.
- `fcontour` plots 2-D contours. Supersedes `ezcontour`.
- `fsurf` plots 3-D surfaces, including parametric surfaces. Supersedes `ezsurf`.
- `fmesh` plots 3-D meshes, including parametric meshes. Supersedes `ezmesh`.

Because the new functions fully integrate with MATLAB graphics, you can use standard MATLAB graphics options as inputs to these functions.

Simscape Component Generation: Create custom components directly from symbolic math equations for use in dynamic simulation

Use `symReadSSCParameters` and `symReadSSCVariables` to load the names, values, and units of the parameters and variables of a Simscape component to MATLAB. Names, values, and units appear in the MATLAB workspace as cell arrays.

When you are ready to import the result of symbolic computations to a Simscape component, use `symWriteSSC`. This function lets you create a new component using an existing component as a template and adding new equations.

MATLAB `cell2sym` and `sym2cell` simplify conversions between symbolic and cell arrays

The MATLAB `cell2sym` function converts cell arrays to symbolic arrays. The MATLAB `sym2cell` function converts symbolic arrays to cell arrays.

MATLAB `nchoosek` accepts a vector as its first argument

`C = nchoosek(v, k)` returns a matrix containing all possible combinations of the elements of vector `v` taken `k` at a time.

MATLAB `sym` creates multidimensional arrays

`sym(a, [n1 ... nM])` creates the symbolic array with dimension `n1`-by-...-by-`nM`. You can create symbolic arrays of any dimension using this syntax. For details, see `sym`.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Character vector inputs to <code>sym</code> and <code>vpa</code> are restricted to numbers and valid variable names.	Warns	<p>When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:</p> <ul style="list-style-type: none"> <code>syms x; x + 1</code> <p>instead of</p> <pre>sym('x + 1')</pre> <ul style="list-style-type: none"> <code>exp(sym(pi))</code> <p>instead of</p> <pre>sym('exp(pi)')</pre> <ul style="list-style-type: none"> <code>syms f(var1,...varN)</code> <p>instead of</p> <pre>f(var1,...varN) = sym('f(var1,...varN)')</pre> <ul style="list-style-type: none"> <code>vpa((1 + sqrt(sym(5)))/2)</code> <p>instead of</p> <pre>vpa('(1 + sqrt(5))/2')</pre>	Support of character vectors that are not valid variable names and do not define a number will be removed in a future release. To create symbolic expressions, first create symbolic variables, and then use operations on them.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Comparing symbolic and non-symbolic objects by using <code>isequal</code>	Returns 0 instead of 1.	Wrap the non-symbolic object with <code>sym</code> . For example, instead of <code>a=1; isequal(a,1)</code> , use <code>isequal(a,sym(1))</code> .	Symbolic objects are not considered equal to non-symbolic objects.
Character vectors as inputs to <code>solve</code>	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>solve('2*r = 1', 'r')</code> with <code>syms r; solve(2*r == 1, r)</code> .	Do not specify equations and variables as character vectors. Instead of character vector inputs, create symbolic variables using <code>syms</code> , and then pass them as input arguments separated by commas, or as a vector of input arguments.
<code>findsym</code>	Warns	<code>symvar</code>	Replace all instances of <code>findsym</code> with <code>symvar</code> .
<code>mfun</code>	Warns	Appropriate special function. For example, replace <code>mfun('P', 1,2,3,4)</code> with <code>jacobiP(1,2,3,4)</code> .	Replace all instances of <code>mfun</code> with the appropriate function call. See Special Functions for the list of available special functions.
<code>mfunlist</code>	Warns	See Special Functions.	See Special Functions for the list of available special functions.
<code>sym(A, set)</code> and <code>sym(A, 'clear')</code> where <code>A</code> is a symbolic object in the workspace.	Errors	<code>assume(A, set)</code> and <code>assume(A, 'clear')</code>	Instead of <code>sym</code> , use <code>assume</code> to set and clear assumptions on variables in the workspace.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Values All and None of IgnoreAnalyticConstraints	Errors	true and false	Replace all instances of 'IgnoreAnalyticConstraints', 'All' with 'IgnoreAnalyticConstraints', true. Replace all instances of 'IgnoreAnalyticConstraints', 'None' with 'IgnoreAnalyticConstraints', false.
poly2sym(c, 'var') does not accept a character vector 'var' anymore.	Errors	syms var; poly2sym(c, var) or poly2sym(c, sym('var'))	Replace all instances of poly2sym(c, 'var') with syms var; poly2sym(c, var) or poly2sym(c, sym('var'))

R2015b

Version: 6.3

New Features

Bug Fixes

Compatibility Considerations

Fourier and Laplace transforms and their inverses for a wider variety of input expressions, including hyperbolic functions

More patterns are available for the transformation functions `fourier`, `laplace`, `ztrans` and their inverses, allowing these functions to support a wider variety of input expressions.

MATLAB series function for computing Puiseux series expansion

The MATLAB `series` function approximates a symbolic expression or function with a Puiseux series expansion.

MATLAB `hermiteForm` and `smithForm` functions for computing Hermite and Smith normal forms of matrices

The MATLAB `smithForm` and `hermiteForm` functions compute the Smith and Hermite normal forms of a matrix, respectively. Elements of a matrix must be integers or polynomials. Both functions also can return corresponding transformation matrices.

The MuPAD `linalg::smithForm` and `linalg::hermiteForm` functions provide more functionality:

- `linalg::smithForm` returns transformation matrices along with the Smith form of a matrix.
- `linalg::hermiteForm` computes the Hermite form of a matrix of polynomials.

Sparse argument for `matlabFunction`, `odeFunction`, and `daeFunction` for using sparse instead of dense matrices in generated MATLAB functions

`matlabFunction`, `odeFunction`, and `daeFunction` accept the name-value pair argument, `'Sparse', true` that triggers the generated MATLAB functions to represent symbolic matrices by sparse numeric matrices in the generated code.

MATLAB has function for searching subexpressions in a symbolic expression

The MATLAB `has` function checks if an expression contains specified subexpressions.

MATLAB `root` function for representing roots of polynomials

The MATLAB `root` function represents roots of polynomials. Symbolically solving a high degree polynomial for its roots can be complex or mathematically impossible. In this case, Symbolic Math Toolbox uses the `root` function to represent the roots of the polynomial.

New Symbolic Math Toolbox examples

The following new examples demonstrate the functionality of Symbolic Math Toolbox:

- Numerical Computations With High Precision. To run this example, enter `NumericComputingWithHighPrecisionExample` in the MATLAB Command Window.
- Decimal Digits of pi. To run this example, enter `DigitsOfPiExample` in the MATLAB Command Window.
- Prime Factorizations. To run this example, enter `PrimesExample` in the MATLAB Command Window.
- Handling Large Integers to Solve the 196 Problem. To run this example, enter `PalindromeExample` in the MATLAB Command Window.
- Rotation Matrix and Transformation Matrix. To run this example, enter `RotationExample` in the MATLAB Command Window.
- Gauss-Laguerre Quadrature Evaluation Points and Weights. To run this example, enter `QuadratureRulesExample` in the MATLAB Command Window.
- Simulate a Stochastic Process by Feynman-Kac Formula. To run this example, enter `FeynmanKacExample` in the MATLAB Command Window.

The following examples are updated and renamed:

- Integration. To run this example, enter `IntExample` in the MATLAB Command Window.

- Differentiation. To run this example, enter `DiffExample` in the MATLAB Command Window.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Character vectors inputs to <code>sym</code> and <code>vpa</code> are restricted to numbers and valid variable names.	Still runs	<p>When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:</p> <ul style="list-style-type: none"> • <code>syms x; x + 1</code> instead of <code>sym('x + 1')</code> • <code>exp(sym(pi))</code> instead of <code>sym('exp(pi)')</code> • <code>syms f(var1,...varN)</code> instead of <code>f(var1,...varN) = sym('f(var1,...varN)')</code> • <code>vpa((1 + sqrt(sym(5)))/2)</code> instead of <code>vpa('(1 + sqrt(5))/2')</code> 	Support of character vectors that are not valid variable names and do not define a number will be removed in a future release. To create symbolic expressions, first create symbolic variables, and then use operations on them.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Character vectors as inputs to <code>solve</code>	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>solve('2*r = 1', 'r')</code> with <code>syms r; solve(2*r == 1, r)</code> .	Do not specify equations and variables as character vectors. Instead of character vector, create symbolic variables using <code>syms</code> , and then pass them as input arguments separated by commas, or as a vector of input arguments.
<code>findsym</code>	Warns	<code>symvar</code>	Replace all instances of <code>findsym</code> with <code>symvar</code> .
Values <code>All</code> and <code>None</code> of <code>IgnoreAnalyticConstraints</code>	Warns	<code>true</code> and <code>false</code>	Replace all instances of <code>'IgnoreAnalyticConstraints', 'All'</code> with <code>'IgnoreAnalyticConstraints', true</code> . Replace all instances of <code>'IgnoreAnalyticConstraints', 'None'</code> with <code>'IgnoreAnalyticConstraints', false</code> .
<code>mfun</code>	Warns	Appropriate special function. For example, replace <code>mfun('P', 1, 2, 3, 4)</code> with <code>jacobiP(1, 2, 3, 4)</code> .	Replace all instances of <code>mfun</code> with the appropriate function call. See Special Functions for the list of available special functions.
<code>mfunlist</code>	Warns	See Special Functions .	See Special Functions for the list of available special functions.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
poly2sym(c, 'var') will not accept a character vector 'var' in a future release.	Warns	syms var; poly2sym(c, var) or poly2sym(c, sym('var'))	Replace all instances of poly2sym(c, 'var') with syms var; poly2sym(c, var) or poly2sym(c, sym('var'))
setVar(nb, 'MuPADvar')	Errors	Three-argument version setVar(nb, 'MuPADvar', MATLABexpr)	Replace all instances of setVar(nb, 'MuPADvar') with setVar(nb, 'MuPADvar', MATLABexpr).

R2015a

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

MATLAB `functionalDerivative` function for finding derivatives of functionals

The MATLAB `functionalDerivative` function finds the derivative of a symbolic expression with respect to functions.

MATLAB `odeFunction` for converting systems of algebraic expressions to MATLAB functions suitable for `ode45` and other ODE solvers

The MATLAB `odeFunction` converts a system of symbolic algebraic expressions to MATLAB function handle suitable for `ode45`, `ode15s`, and other ODE solvers.

MATLAB `partfrac` function for computing partial fraction decomposition

The MATLAB `partfrac` function finds the partial fraction decomposition of a rational expression. This function accepts the name-value pair argument `'FactorMode'`, mode that lets you choose a factorization mode used to factor the denominator. Here, mode is one of the following: `'real'`, `'complex'`, `'full'`, or `'rational'`. By default, `partfrac` performs factorization over the rational numbers.

MATLAB `sympref` function for specifying preferences for symbolic functions `fourier`, `ifourier`, and `heaviside`

The MATLAB `sympref` function specifies preferences for symbolic functions `fourier`, `ifourier`, and `heaviside`. `sympref` specifies values of parameters in `fourier` and `ifourier`, and the return value of `heaviside` at 0.

Optimize argument for controlling code optimization in generated MATLAB functions returned by `matlabFunction`, `odeFunction`, and `daeFunction`

`matlabFunction`, `odeFunction`, and `daeFunction` accept the name-value pair argument, `'Optimize'`, `false` that disables code optimization when you write the resulting code to a file.

MuPAD isolate function for rearranging an equation so that the variable or expression appears on the left side

The MuPAD `isolate` function rearranges an equation so that the variable or expression appears on the left side of the equation. If `isolate` cannot isolate the variable or expression, it moves all terms containing the variable or expression to the left side.

FactorMode argument offering different modes of factorization returned by MATLAB factor function

The MATLAB `factor` function now accepts the name-value pair argument `'FactorMode', mode` that lets you choose a factorization mode. Here, `mode` is one of the following: `'real'`, `'complex'`, `'full'`, or `'rational'`. By default, `factor` performs factorization over the rational numbers.

Reverse accumulation option for cumsum and cumprod functions

The `'reverse'` option for the MATLAB `cumsum` and `cumprod` functions reverses the direction of cumulation, working from end to 1 in the active dimension. This option allows quick directional calculations without requiring a flip or reflection of the input array.

MATLAB functions chol, lu, qr, and rank now return certain outputs as type double

The MATLAB functions `chol`, `lu`, `qr`, and `rank` now return certain outputs as type `double` instead of symbolic objects.

Compatibility Considerations

The functions `chol`, `lu`, and `qr` now return the permutation information as matrices or vectors of double-precision values. The `rank` function returns the rank of a matrix as a double-precision value. In previous releases, these output arguments were returned as symbolic objects. For details, see the Output Arguments section on the respective reference pages.

MATLAB functions `assume`, `assumeAlso`, `assumptions`, `sym`, and `syms` have changes to assumptions mechanism

- `syms` creates a vector or a matrix of symbolic variables where each element of the vector or matrix belongs to `set` using the syntax `sym(A, dim, set)`. For example, `A = sym('A', [3 3], 'rational')` creates the 3-by-3 matrix `A` where MATLAB assumes all elements of `A` are rational.
- `assume` clears assumptions on a variable `var` using the syntax `assume(var, 'clear')`. If `var` is an expression, `assume` clears all assumptions on all variables in `var`.
- `assume` and `assumeAlso` set the assumption that a variable is positive using the option `'positive'`. For example, assume `x` is positive using `assume(x, 'positive')`.

Compatibility Considerations

- `syms` does not create variables with the following names: `clear`, `integer`, `positive`, `rational`, and `real`. For example, in previous releases `syms integer` created the symbolic variable `integer`. To use these variable names now, use `sym`. For example, to create the symbolic variable `integer`, use `integer = sym('integer')`.
- The syntax `sym(x, set)` for `x` that already exists in the MATLAB workspace will be removed in a future release. Use `assume(x, set)` instead.
- The syntax `sym(x, 'clear')` will be removed in a future release. Use `assume(x, 'clear')` instead.

MATLAB function `combine` combines additional expressions

The MATLAB `combine` can combine expressions containing the function `int` using the target `int`, and expressions containing a sum of sine or cosine functions using the target `sincos`.

New and updated Symbolic Math Toolbox examples

The following new examples demonstrate the functionality of Symbolic Math Toolbox.

-
- **Explore Single-Period Asset Arbitrage.** To run this example, enter `ArbitrageExample` in the MATLAB Command Window.
 - **Compute Binomial Coefficients Exactly.** To run this example, enter `BinomialCoefficientExample` in the MATLAB Command Window.
 - **Electric Dipole Moment and Radiation Power.** To run this example, enter `DipoleExample` in the MATLAB Command Window.
 - **Harmonic Analysis of Transfer Function Output.** To run this example, enter `HarmonicFrequencyExample` in the MATLAB Command Window.
 - **Hilbert Matrices and Their Inverses.** To run this example, enter `HilbertMatrixExample` in the MATLAB Command Window.
 - **Markov Chain Analysis and Stationary Distribution.** To run this example, enter `MarkovChainExample` in the MATLAB Command Window.
 - **Solve a Second-Order Differential Equation Numerically.** To run this example, enter `OdeToVectorFieldExample` in the MATLAB Command Window.
 - **Padé Approximant of Time Delay Input.** To run this example, enter `PadeApproximantExample` in the MATLAB Command Window.

The Differentiation example is updated and renamed. To run this example, enter `DiffExample` in the MATLAB Command Window.

Compatibility Considerations

To run the Differentiation example in previous releases, enter `symsdiffdemo` in the MATLAB Command Window.

MATLAB solve function uses default MaxDegree value of 2

The MATLAB `solve` function uses a default `MaxDegree` value of 2. In previous releases, the default value of `MaxDegree` was 3.

Compatibility Considerations

In previous releases, `solve` automatically returned explicit solutions for equations of degree 3. To obtain the same results as in previous releases, set `MaxDegree` to 3. For example, `solve(a*x^3 + b*x^2 + c*x + 1, 'MaxDegree', 3)`.

MuPAD functions `taylor` and `mtaylor` error when they cannot find a Taylor series

The MuPAD functions `taylor` and `mtaylor` throw an error when they cannot find a Taylor series.

Compatibility Considerations

In previous releases, `taylor` and `mtaylor` issued a warning and returned unresolved symbolic `taylor` and `mtaylor` calls in such cases.

MuPAD orthogonal polynomial functions return polynomial expressions

The MuPAD orthogonal polynomial functions return polynomial expressions when the polynomials are evaluated with identifiers. This applies to all functions in the MuPAD `orthpoly` library.

Compatibility Considerations

In previous releases, the MuPAD orthogonal polynomial functions returned objects of type `DOM_POLY` when the polynomials were evaluated with identifiers.

MATLAB function `sym` treats `i` in character vectors as a variable

The MATLAB function `sym` treats `i` in character vectors as a variable. For example, `sym('1 + i')` returns the symbolic expression $i+1$.

Compatibility Considerations

- In previous releases, `sym` treated `i` in character vectors as an imaginary number. Now, it is treated as a variable `i`. For example, `sym('1 + i')^2` returns the symbolic expression $(i + 1)^2$. To obtain the same results as in previous releases, specify the imaginary number `i` as `1i`. For example, `sym('1 + 1i')^2` returns $2i$.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>findsym</code>	Still runs	<code>symvar</code>	Replace all instances of <code>findsym</code> with <code>symvar</code> .
Values <code>All</code> and <code>None</code> of <code>IgnoreAnalyticConstraints</code>	Still runs	<code>true</code> and <code>false</code>	Replace all instances of <code>'IgnoreAnalyticConstraints', 'All'</code> with <code>'IgnoreAnalyticConstraints', true</code> . Replace all instances of <code>'IgnoreAnalyticConstraints', 'None'</code> with <code>'IgnoreAnalyticConstraints', false</code> .
<code>mfun</code>	Warns	Appropriate special function. For example, replace <code>mfun('P', 1, 2, 3, 4)</code> with <code>jacobiP(1, 2, 3, 4)</code> .	Replace all instances of <code>mfun</code> with the appropriate function call. See Special Functions for the list of available special functions.
<code>mfunlist</code>	Warns	See Special Functions.	See Special Functions for the list of available special functions.
<code>setVar(nb, 'MuPADvar')</code>	Warns	Three-argument version <code>setVar(nb, 'MuPADvar', MATLABexpr)</code>	Replace all instances of <code>setVar(nb, 'MuPADvar')</code> with <code>setVar(nb, 'MuPADvar', MATLABexpr)</code>

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Character vectors as inputs to <code>solve</code>	Warns	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace <code>solve('2*r = 1, r')</code> with <code>syms r; solve(2*r == 1, r)</code> .	Do not specify equations and variables as character vectors. Instead of character vectors, create symbolic variables using <code>syms</code> , and then pass them as input arguments separated by commas, or as a vector of input arguments.
<code>poly2sym(c, 'var')</code> will not accept a character vector 'var' in a future release.	Warns	<code>syms var;</code> <code>poly2sym(c, var)</code> or <code>poly2sym(c, sym('var'))</code>	Replace all instances of <code>poly2sym(c, 'var')</code> with <code>syms var; poly2sym(c, var)</code> or <code>poly2sym(c, sym('var'))</code>
<code>simple</code>	Errors	<code>simplify</code>	Replace all instances of <code>simple(S)</code> with <code>simplify(S)</code> . There is no replacement for <code>[r, how] = simple(S)</code> .
In previous releases, <code>in(x, type)</code> in some cases returned logical 1 if <code>x</code> belonged to <code>type</code> and 0 otherwise.	Returns a symbolic expression of the form <code>in(x, type)</code>	<code>isAlways</code>	To obtain the same results as in previous releases, wrap such expressions in <code>isAlways</code> . For example, use <code>isAlways(in(sym(5), 'integer'))</code> .
In previous releases, the symbolic relational operators in some cases evaluated equations involving only symbolic numbers and returned logical 1 or 0.	Returns a symbolic equation or inequality	<code>isAlways</code>	To obtain the same results as in previous releases, wrap equations in <code>isAlways</code> . For example, use <code>isAlways(A < B)</code> .

R2014b

Version: 6.1

New Features

Bug Fixes

Compatibility Considerations

MATLAB solve function returning parameters and conditions in solutions

The MATLAB symbolic `solve` function returns the parameters in a solution and the conditions under which a solution is valid when the `ReturnConditions` option is specified as `true`.

Compatibility Considerations

- Do not specify equations and variables as character vectors. Instead of character vectors, declare symbolic variables using `syms`, and then pass them as input arguments separated by commas, or as a vector of input arguments. For example, replace

```
solve('2*r = 1, r')
```

with

```
syms r  
solve(2*r == 1, r)
```

In a future release, character vector input arguments will be interpreted as option names to support shortcuts, such as `ignorea` for `IgnoreAnalyticConstraints`.

- `solve` warns when it calls the numerical solver and returns a numerical output.
- `solve` does not warn if provably no solution exists.
- When the list of equations is empty, `solve` throws an error instead of a warning.

Functions for analyzing and reducing systems of differential algebraic equations (DAEs), such as `isLowIndexDAE` and `reduceDAEIndex`

These MATLAB and MuPAD functions help you

- Identify subsets (blocks) of equations that can be used to define subsets of variables.
- Identify high-index differential algebraic equations.
- Reduce high-index differential algebraic equations to systems of differential index 1 or 0.
- The MATLAB symbolic `reduceDifferentialOrder` function reduces higher-order differential equations to a system of first-order differential equations. The

`daetools::reduceDifferentialOrder` function provides the same functionality in MuPAD.

- `incidenceMatrix` computes the incidence matrix of a system of differential algebraic equations. The `daetools::incidenceMatrix` function provides the same functionality in MuPAD.
- `reduceRedundancies` eliminates simple equations from a system of symbolic differential algebraic equations. The `daetools::reduceRedundancies` function provides the same functionality in MuPAD.
- `findDecoupledBlocks` searches for decoupled blocks in systems of equations. The `daetools::findDecoupledBlocks` function provides the same functionality in MuPAD.
- `isLowIndexDAE` tests if a first-order system of differential algebraic equations is of differential index 0 or 1. The `daetools::isLowIndexDAE` function provides the same functionality in MuPAD.
- `reduceDAEIndex` converts a system of first-order differential algebraic equations to an equivalent system of differential index 1. The `daetools::reduceDAEIndex` function provides the same functionality in MuPAD.
- `reduceDAEToODE` reduces the differential index of a system of first-order semilinear differential algebraic equations to 0. The `daetools::reduceDAEToODE` function provides the same functionality in MuPAD.
- `daeFunction` converts a system of differential algebraic equations to a MATLAB function handle.
- `massMatrixForm` extracts the mass matrix and the right sides of a semilinear system of symbolic differential algebraic equations. The `daetools::massMatrixForm` function provides the same functionality in MuPAD.
- `decic` computes consistent initial conditions for `ode15i`.

MATLAB functions representing orthogonal polynomials: `chebyshevT`, `chebyshevU`, `legendreP`, `laguerreL`, `hermiteH`, `jacobiP`, and `gegenbauerC`

- The MATLAB symbolic `chebyshevT` and `chebyshevU` functions represent Chebyshev polynomials of the first and second kind, respectively.
- The MATLAB symbolic `gegenbauerC` function represents Gegenbauer polynomials.

- The MATLAB symbolic `hermiteH` function represents Hermite polynomials.
- The MATLAB symbolic `jacobiP` function represents Jacobi polynomials.
- The MATLAB symbolic `legendreP` function represents Legendre polynomials.
- The MATLAB symbolic `laguerreL` function represents Laguerre polynomials.

MATLAB `pade` function for computing Padé approximation

The MATLAB symbolic `pade` function calculates Padé approximations.

MATLAB `funm` function for computing matrix functions

The MATLAB symbolic `funm` function and the MuPAD `funm` function represent a general matrix function. A matrix function is a scalar function that maps one matrix to another, for example, function f in $B = f(A)$ maps matrix A to matrix B .

MATLAB `kummerU` function for computing confluent hypergeometric (Kummer U) function

The MATLAB symbolic `kummerU` function computes the value of the confluent hypergeometric function. This function is also known as the Kummer U function.

MATLAB `polylog` function for computing polylogarithms

The MATLAB symbolic `polylog` function computes polylogarithms.

MATLAB `signIm` function for computing signs of imaginary parts of complex numbers

The MATLAB symbolic `signIm` function returns a sign of the imaginary part of a complex number. For all complex numbers with a nonzero imaginary part, $\text{signIm}(z) = \text{sign}(\text{imag}(z))$. For real numbers, $\text{signIm}(z) = -\text{sign}(z)$.

MATLAB `in` function for representing conditions on symbolic inputs

The MATLAB symbolic `in` function represents the condition that the input is of the specified type. The allowed types are `integer`, `real` and `rational`. The `in` function is

used in the input and output of other functions such as `solve` to represent conditions on symbolic variables. If the input is a number of the specified type, the `in` function returns logical 1 (`true`), and if it is not of the specified type, the `in` function returns logical 0 (`false`).

MATLAB divisors function for finding divisors of integers and polynomials

The MATLAB symbolic `divisors` function computes divisors of integers and polynomials.

MATLAB functions `nnz` and `nonzeros` for finding nonzero elements in a symbolic array

The MATLAB symbolic `nnz` function computes the number of nonzero elements in a symbolic vector, matrix, or multidimensional array.

The MATLAB symbolic `nonzeros` function returns a column vector containing all nonzero elements of a symbolic vector, matrix, or multidimensional array.

MATLAB `pochhammer` function to calculate Pochhammer symbol

The MATLAB symbolic `pochhammer` function calculates the Pochhammer symbol.

MATLAB `kronckerDelta` function for computing the Kronecker delta function

The MATLAB symbolic `kronckerDelta` function calculates the Kronecker delta function.

MATLAB `dirac` function with two input arguments for computing derivatives of the Dirac delta function

The MATLAB symbolic `dirac` function with one input argument represents the Dirac delta function. `dirac` with two input arguments, `dirac(n, x)` represents the n th derivative of the Dirac delta function at x .

MATLAB `isAlways` function warns when returning false for undecidable inputs

The MATLAB symbolic `isAlways` function issues a warning when it returns logical 0 (`false`) for an undecidable input.

Compatibility Considerations

- In previous releases, `isAlways` did not issue a warning before returning logical 0 (`false`) for an undecidable input. To go back to this behavior, suppress the warning by specifying the `Unknown` option as `false`, as in `isAlways(cond, 'Unknown', 'false')`.

MuPAD `generate::fortran` function can use Fortran 90 as the target compiler

The MuPAD `generate::fortran` function can use Fortran 90 as the target compiler, in addition to Fortran 77.

MATLAB `mod` function for computing modulus after division

The MATLAB symbolic `mod` function finds the modulus after division as follows:
 $\text{mod}(a,b) = a - b \cdot \text{floor}(a/b)$.

Compatibility Considerations

The MATLAB symbolic `mod` function uses the same definition as the MuPAD `modp` function. Also, by default, the MuPAD `mod` operator and its functional form `_mod` are equivalent to the MuPAD `modp` function.

In previous releases, the MuPAD `modp` and `mods` functions computed the modulus after division according to these definitions:

- If a and b are integers, then `modp(a,b)` is an integer r , such that $a = qb + r$, $0 \leq r < |b|$, and $q = a \text{ div } b$.

If b is an integer and a is a rational number, $a = u/v$, such that v and b are coprime integers, then $\text{modp}(a, b) = \text{modp}(u*w, b)$. Here, w is an inverse of $v \pmod b$, that is, $v*w \equiv 1 \pmod b$.

- If a and b are integers, then $\text{mods}(a, b)$ is an integer r , such that $a = qb + r$ and $-|b|/2 < r \leq |b|/2$.

If b is an integer and a is a rational number, $a = u/v$, such that v and b are coprime integers, then $\text{mods}(a, b) = \text{mods}(u*w, b)$. Here, w is an inverse of $v \pmod b$, that is, $v*w \equiv 1 \pmod b$.

Now, MuPAD uses the same definitions for an integer or rational a . As in previous releases, b must be an integer.

- $\text{modp}(a, b) = a - b*\text{floor}(a/b)$
- $\text{mods}(a, b) = a - b*\text{round}(a/b)$

In MuPAD, to get the same results as in previous releases, use the MuPAD `numlib::lincongruence` function to find a modular inverse, and then use the MuPAD modulo operator or functions, for example:

```
w := numlib::lincongruence(3, 1, 5)[1]:
22*w mod 5;
modp(22*w, 5);
mods(22*w, 5);
_mod(22*w, 5)
```

In MATLAB, to get the same results as in previous releases, use the `gcd` function with three output arguments to find a modular inverse, and then use the `mod` function as follows. For example, for $\text{mod}(a, b) = \text{mod}(u/v, b)$ use these commands:

```
[~, A, ~] = gcd(sym(v), b);
mod(A*u, b)
```

MATLAB gcd and lcm functions accept vectors and matrices

The MATLAB symbolic `gcd` and `lcm` functions accept vectors and matrices as input arguments. If A is a vector or a matrix, then `gcd(A)` and `lcm(A)` find the greatest common divisor and least common multiple of all elements of A . If A and B are vectors or matrices of the same size, then `gcd(A, B)` and `lcm(A, B)` find the greatest common divisor and least common multiple of the pairs of elements of A and B .

MATLAB rem function accepts vectors and matrices

The MATLAB symbolic `rem` function accepts vectors and matrices as input arguments. Also, the new MuPAD `rem` function lets you compute a remainder after division in a MuPAD notebook.

Compatibility Considerations

In previous release, `rem` accepted polynomials as its input arguments. For example, `sym x; R = rem(x^2 + 2, x)` returned 2. To perform polynomial division in this and later releases, use the `quorem` function, for example, `sym x; [~, R] = quorem(x^2 + 2, x)`.

MATLAB factor function only accepts scalar inputs and returns vector of factors of input

The MATLAB symbolic `factor` function only accepts scalar inputs. The `factor` function returns a symbolic vector of irreducible factors of the input.

Compatibility Considerations

The `factor` function does not accept nonscalar inputs. The output is a symbolic vector and is not of type `factored`.

MuPAD Notebook app supports left and right double square brackets

The MuPAD `Symbol::LeftDoubleBracket` and `Symbol::RightDoubleBracket` functions insert left and right double square brackets. These symbols are also called white square brackets.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>mfun</code>	Still runs	Appropriate special function. For example, replace <code>mfun('P', 1, 2, 3, 4)</code> with <code>jacobiP(1, 2, 3, 4)</code> .	Replace all instances of <code>mfun</code> with the appropriate function call. See Special Functions for the list of available special functions.
<code>mfunlist</code>	Still runs	See Special Functions.	See Special Functions for the list of available special functions.
<code>setVar(nb, 'MuPADvar')</code>	Warns	Three-argument version <code>setVar(nb, 'MuPADvar', MATLABexpr)</code>	Replace all instances of <code>setVar(nb, 'MuPADvar')</code> with <code>setVar(nb, 'MuPADvar', MATLABexpr)</code>
<code>simple</code>	Warns	<code>simplify</code>	Replace all instances of <code>simple(S)</code> with <code>simplify(S)</code> . There is no replacement for <code>[r, how] = simple(S)</code> .
<code>emlBlock</code>	Errors	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .

R2014a

Version: 6.0

New Features

Bug Fixes

Compatibility Considerations

MATLAB functions for computing special integrals, gamma functions, dilogarithm function, and number-theoretic functions

The following special functions are available:

- The MATLAB symbolic `sinhint` and `coshint` function compute the hyperbolic sine and cosine integral functions, respectively.
- The MATLAB symbolic `ssinint` function computes the shifted sine integral function.
- The MATLAB symbolic `dawson` function computes the Dawson integral.
- The MATLAB symbolic `fresnelc` and `fresnels` functions return the Fresnel cosine and sine integral functions respectively.
- The MATLAB symbolic `logint` function computes the logarithmic integral function. This function is also called the integral logarithm.
- The MATLAB symbolic `gammaln` function computes the logarithmic gamma function.
- The MATLAB symbolic `igamma` function computes the incomplete gamma function.
- The MATLAB symbolic `dilog` function computes the dilogarithm function.
- The MATLAB symbolic `bernoulli` function computes the Bernoulli numbers and polynomials.
- The MATLAB symbolic `euler` function computes the Euler numbers and polynomials.
- The MATLAB symbolic `harmonic` function computes the harmonic function. For positive integer arguments, the harmonic function produces harmonic numbers.
- The MATLAB symbolic `catalan` function represents the Catalan constant. To approximate the Catalan constant with the current precision set by `digits`, use `vpa(catalan)`.
- The MATLAB symbolic `eulergamma` function represents the Euler-Mascheroni constant. To approximate the Euler-Mascheroni constant with the current precision set by `digits`, use `vpa(eulergamma)`.

MATLAB function `qr` for computing symbolic QR factorization

The MATLAB symbolic `qr` function computes the QR factorization of a matrix. The result can be used to solve matrix equations.

MATLAB function combine for combining symbolic expressions with multiple calls to the same function

The MATLAB symbolic `combine` function applies rewriting rules to the input expression to combine multiple calls to a function, and returns the rewritten expression. The analytic constraints on applying rewriting rules can be optionally relaxed when the function is called.

MATLAB functions max and min for finding the largest and smallest elements of a symbolic array

The MATLAB symbolic `max` and `min` functions return the largest and the smallest element of a symbolic vector or matrix, all elements of which are convertible to floating-point numbers. For a symbolic matrix, these functions find the largest and smallest elements of each row or column.

vpasolve can use random starting points when searching for solutions

The MATLAB numeric solver `vpasolve` now uses random starting points when searching for solutions if you specify `random`. This enables the solver to find different solutions for nonpolynomial equations in subsequent calls.

Support for Unicode characters in MuPAD that includes using Asian language characters in character vectors and text

The toolbox provides support for Unicode® characters in MuPAD (including messages to print or display), variable names, file names, and external file content.

Compatibility Considerations

In previous releases, the MuPAD `strmatch` function used `^[]` to match any characters excluding `.` For example, the command `strmatch("a[b", "^[]", All)` returned `{"a", "b"}`.

Now, use `^[\\]` to match any characters excluding `.` Thus, rewrite the example as follows: `strmatch("a[b", "^[\\]", All)`.

`strmatch` requires the same change for the closing parenthesis `]`.

Support for specifying encoding in MuPAD file operations

The MuPAD functions for file operations, such as `finput`, `fopen`, `fprint`, `read`, `write` and more, accept the option `Encoding`. This option lets you specify the following values for encoding.

Big5	ISO-8859-1	windows-932
EUC-JP	ISO-8859-2	windows-936
GBK	ISO-8859-3	windows-949
KSC_5601	ISO-8859-4	windows-950
Macintosh	ISO-8859-9	windows-1250
Shift_JIS	ISO-8859-13	windows-1251
US-ASCII	ISO-8859-15	windows-1252
UTF-8		windows-1253
		windows-1254
		windows-1257

Choice of right- or left-handed spherical coordinate system for the MuPAD vector analysis functions

The MuPAD vector analysis functions `curl`, `divergence`, `gradient`, `laplacian`, and `linalg::ogCoordTab` let you choose between right- and left-handed spherical coordinate systems. By default, these functions use the right-handed coordinate system with `[radial, polar, azimuthal]` coordinates. To switch to `[radial, azimuthal, polar]` coordinates, specify `Spherical[LeftHanded]`.

Compatibility Considerations

In previous releases, the MuPAD vector analysis functions used the left-handed spherical coordinate system. To get the same results as in previous releases, use `'Spherical[LeftHanded]'`. To use the right-handed spherical coordinate system and suppress the warning, use `'Spherical[RightHanded]'`.

MATLAB special functions and functions for computing integral and Z-transforms accept several nonscalar arguments

The following MATLAB symbolic functions now accept more than one nonscalar argument:

- `airy` representing the Airy function
- `besseli`, `besselj`, `bessely`, and `besselk` representing the Bessel functions of the first and second kind, and the modified Bessel functions
- `beta` representing the beta function
- `ellipticE`, `ellipticF`, `ellipticPi`, and `ellipticCPi` representing the elliptic integrals
- `lambertw` representing the Lambert W function
- `whittakerM` and `whittakerW` representing the Whittaker M and Whittaker W functions
- `psi` representing the polygamma function
- `fourier` and `ifourier` representing the Fourier and inverse Fourier transforms
- `laplace` and `ilaplace` representing the Laplace and inverse Laplace transforms
- `ztrans` and `iztrans` representing the Z-transform and inverse Z-transform

MATLAB function `erfc` accepts two arguments

The MATLAB symbolic `erfc` function with one input argument represents the complementary error function. `erfc` with two input arguments represents the iterated integrals of the complementary error function, $\text{erfc}(k, x) = \int (\text{erfc}(k - 1, y), y, x, \text{inf})$.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD <code>linalg::curl</code>	Still runs	<code>curl</code>	Replace all instances of <code>linalg::curl</code> with <code>curl</code> .
MuPAD <code>linalg::det</code>	Still runs	<code>det</code>	Replace all instances of <code>linalg::det</code> with <code>det</code> .
MuPAD <code>linalg::divergence</code>	Still runs	<code>divergence</code>	Replace all instances of <code>linalg::divergence</code> with <code>divergence</code> .
MuPAD <code>linalg::grad</code>	Still runs	<code>gradient</code>	Replace all instances of <code>linalg::grad</code> with <code>gradient</code> .
MuPAD <code>linalg::gradient</code>	Still runs	<code>gradient</code>	Replace all instances of <code>linalg::gradient</code> with <code>gradient</code> .
MuPAD <code>linalg::hessian</code>	Still runs	<code>hessian</code>	Replace all instances of <code>linalg::hessian</code> with <code>hessian</code> .
MuPAD <code>linalg::jacobian</code>	Still runs	<code>jacobian</code>	Replace all instances of <code>linalg::jacobian</code> with <code>jacobian</code> .
MuPAD <code>linalg::laplacian</code>	Still runs	<code>laplacian</code>	Replace all instances of <code>linalg::laplacian</code> with <code>laplacian</code> .
MuPAD <code>linalg::potential</code>	Still runs	<code>potential</code>	Replace all instances of <code>linalg::potential</code> with <code>potential</code> .
MuPAD <code>linalg::vectorPotential</code>	Still runs	<code>vectorPotential</code>	Replace all instances of <code>linalg::vectorPotential</code> with <code>vectorPotential</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>simple</code>	Warns	<code>simplify</code>	Replace all instances of <code>simple(S)</code> with <code>simplify(S)</code> . There is no replacement for <code>[r, how] = simple(S)</code> .
<code>emlBlock</code>	Errors	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .

R2013b

Version: 5.11

New Features

Bug Fixes

Compatibility Considerations

MATLAB `evaluateMuPADNotebook` and `allMuPADNotebooks` functions to evaluate MuPAD notebooks and return list of open notebooks

The MATLAB symbolic `evaluateMuPADNotebook` function lets you evaluate a MuPAD notebook from MATLAB without leaving the MATLAB Command Window or MATLAB Editor. You also can interrupt an evaluation of a MuPAD notebook from MATLAB.

The MATLAB symbolic `allMuPADNotebooks` function identifies all currently open notebooks and returns a vector of handles to them. You can use this vector to evaluate all or some of the notebooks or close them. If you already created a MuPAD notebook without a handle or if you lost the handle to a notebook, `allMuPADNotebooks` helps you create a new handle without saving the notebook.

bernstein function for approximating functions using Bernstein polynomials, and `bernsteinMatrix` function for computing Bezier curves

The MATLAB symbolic `bernstein` function and the MuPAD `bernstein` function approximate symbolic expressions and functions by Bernstein polynomials. The MATLAB symbolic `bernsteinMatrix` function and the MuPAD `bernsteinMatrix` function serve for constructing Bezier curves.

MATLAB `cumsum` and `cumprod` functions for computing cumulative sums and products

The MATLAB symbolic `cumsum` and `cumprod` functions return cumulative sums and products of elements of symbolic vectors and matrices.

MATLAB `isfinite`, `isinf`, and `isnan` functions for testing for finite, infinite, and NaN elements in symbolic arrays

The MATLAB symbolic `isfinite`, `isinf`, and `isnan` functions test whether the elements of a symbolic array are finite, infinite, or NaNs.

ExclusiveConditions option that makes MuPAD piecewise function equivalent to an `if-elif-end_if` statement

The new `ExclusiveConditions` option of the MuPAD piecewise function fixes the order of branches in a piecewise expression. Thus, `piecewise` with `ExclusiveConditions` is

almost equivalent to an `if-elif-end_if` statement, except that `piecewise` takes into account assumptions on identifiers. For example, if the condition in the first branch returns `TRUE`, then `piecewise` returns the expression from the first branch. If a true condition appears in any further branch, then `piecewise` returns the expression from that branch and removes all subsequent branches.

MATLAB `mupadNotebookTitle` function to find the window title of the MuPAD notebook

The MATLAB symbolic `mupadNotebookTitle` function returns a cell array containing the window title of the MuPAD notebook. This function lets you find the title of a particular notebook as well as all currently open notebooks.

MATLAB `close` function to close MuPAD notebooks from MATLAB

The MATLAB symbolic `close` function lets you close MuPAD notebooks without leaving the MATLAB Command Window. This function also accepts the `'force'` flag suppressing the dialog box that prompts you to save changes.

`diff` supports mixed derivatives

The MATLAB symbolic `diff` function lets you compute mixed derivatives in one function call. For example, `diff(S, x, y)` differentiates the expression `S` with respect to the variables `x`, and then differentiates the result with respect to the variable `y`.

`coeffs` function extracts coefficients of multivariate polynomials

The MATLAB symbolic `coeffs` function returns coefficients of multivariate polynomials. You can specify polynomial variables as a vector of these variables. If you do not specify the polynomial variables, then `coeffs` regards all symbolic variables found in the polynomial expression as polynomial variables.

linspace, logspace, and compan functions for symbolic objects

The MATLAB `linspace` and `logspace` functions, which generate linearly and logarithmically spaced vectors, and the `compan` function, which finds the companion matrix, now accept symbolic numbers, variables, expressions, and functions.

Indexing uses lists, vectors, and matrices of indices

The MuPAD `_index` function and its equivalent `[]` now accept lists, vectors, and matrices as indices.

MuPAD lets you set assumptions on matrices

The MuPAD `assume`, `assumeAlso`, `assuming`, and `assumingAlso` functions let you set assumptions on matrices.

int, symprod, and symsum let you specify lower and upper bounds as vectors

The MATLAB symbolic `int`, `symprod`, and `symsum` functions accept integration, summation, and product intervals specified by row and column vectors. For example, `int(expr, var, [a, b])`, `int(expr, var, [a b])`, and `int(expr, var, [a; b])` are equivalent to `int(expr, var, a, b)`.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>simple</code>	Still runs	<code>simplify</code>	Replace all instances of <code>simple(S)</code> with <code>simplify(S)</code> . There is no replacement for <code>[r, how] = simple(S)</code> .
<code>emlBlock</code>	Warns	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
diff and int methods for inputs of the char type	Errors	sym	Use the sym method instead.
MuPAD factoring functions numlib::mpqs, numlib::pollard, and numlib::ecm	Errors	ifactor	Replace all instances of numlib::mpqs, numlib::pollard, and numlib::ecm with ifactor.
MuPAD Dom::SparseMatrix F2 domain	Errors	Dom::Matrix(Dom::IntegerMod(2))	Replace all instances of Dom::SparseMatrixF2 with Dom::Matrix(Dom::IntegerMod(2)).
MuPAD userinfo	Errors	print	Use print instead of userinfo.
MuPAD setuserinfo	Errors	prog::trace or debug	Try using prog::trace or debug instead of setuserinfo.

R2013a

Version: 5.10

New Features

Bug Fixes

Compatibility Considerations

Linear algebra functions for computing matrix factorizations (lu, chol), pseudoinverse, orthogonal basis, and adjoint

- `lu` computes the LU factorization of a matrix. Permutation information can be returned as a matrix or as a row vector.
- `chol` computes the Cholesky factorization of a matrix. The result can be returned as an upper or lower triangular matrix. Permutation information can be returned as a matrix or as a row vector.
- `pinv` computes the Moore-Penrose pseudoinverse of a matrix.
- `orth` computes an orthonormal basis for the range of a symbolic matrix.
- `adjoint` computes the adjoint of a symbolic square matrix.

Verification of solutions of systems of equations and arbitrary symbolic function substitution in subs function

The MATLAB symbolic `subs` function lets you:

- Verify solutions of systems of equations by substituting the solutions returned by `solve` back into the systems
- Substitute elements of a symbolic expression with arbitrary symbolic functions

Compatibility Considerations

`subs(s, old, new, 0)` will not accept 0 in a future release. Replace all instances of `subs(s, old, new, 0)` with `subs(s, old, new)`. The `subs` function does not switch `old` and `new` anymore.

`subs` does not return double-precision floating-point results anymore. Instead, it consistently returns symbolic results. To convert such results to double-precision numbers, use `double`.

Simplification for more types of trigonometric and hyperbolic expressions and expressions with nested roots

The MATLAB symbolic `simplify` function and the MuPAD `simplify` function achieve better simplification of trigonometric expressions and expressions with nested roots.

The MATLAB symbolic `simplify` function accepts the new `Criterion` option. This option lets you discourage `simplify` from returning results containing complex numbers.

The MuPAD `simplify` function accepts two new options:

- `Steps` specifies the number of internal simplification steps.
- `Seconds` limits the time allowed for the internal simplification process.

Compatibility Considerations

The default number of simplification steps used by the MATLAB symbolic `simplify` function and the MuPAD `simplify` function changed from 100 to 1.

The `FinalValuation` option used in MuPAD `Simplify` function calls is renamed. The new name is `Criterion`.

Special functions for computing polar angle, `atan2` function, imaginary error function, and exponential and elliptic integrals

- `angle` computes the polar angle of a complex value.
- `atan2` computes the four-quadrant inverse tangent (arctangent).
- `erfi` computes the imaginary error function.
- `ei` computes the one-argument exponential integral.
- `expint` computes the two-argument exponential integral.

The following new MATLAB symbolic functions compute elliptic integrals:

- `ellipticK` computes the complete elliptic integral of the first kind.
- `ellipticF` computes the incomplete elliptic integral of the first kind.
- `ellipticE` computes the complete and incomplete elliptic integrals of the second kind.
- `ellipticCK` computes the complementary complete elliptic integral of the first kind.
- `ellipticCE` computes the complementary complete elliptic integral of the second kind.

- `ellipticPi` computes the complete and incomplete elliptic integrals of the third kind.
- `ellipticCPi` computes the complementary complete elliptic integral of the third kind.
- `ellipke` computes the complete elliptic integrals of the first and second kinds simultaneously.

toeplitz function for creating Toeplitz matrices

The new MATLAB symbolic `toeplitz` function generates a symbolic Toeplitz matrix from two vectors that specify its first column and first row. This function can also generate a symmetric Toeplitz matrix from one vector.

The MuPAD `linalg::toeplitz` function now generates a Toeplitz matrix from two vectors that specify its first column and first row. (In MuPAD, vectors are created as 1-by- n or n -by-1 matrices.) `linalg::toeplitz` accepts the new syntaxes along with the existing syntaxes.

sqrtm function for computing square roots of matrices

The MATLAB symbolic `sqrtm` function computes the square root of a symbolic matrix.

sign function for computing signs of numbers

The MATLAB symbolic `sign` function returns signs of symbolic real and complex values. The sign of a complex value z is defined as $z/abs(z)$.

Real option of the linalg::orthog function for avoiding complex conjugates

The MuPAD `linalg::orthog` function accepts the new `Real` option. This option lets you avoid using a complex scalar product in the orthogonalization process.

Real option of the `linalg::factorCholesky` function for avoiding complex conjugates

The MuPAD `linalg::factorCholesky` function accepts the new `Real` option. When you use this option, `linalg::factorCholesky` assumes that the input matrix is real and symmetric, and does not apply complex conjugation in the course of the algorithm.

Compatibility Considerations

`linalg::factorCholesky` can now compute the Cholesky factorization of a complex Hermitian positive definite matrix. In previous releases, `linalg::factorCholesky` required the input matrix to be symmetric even when working with complex entries. To get the same results as in previous releases for symmetric matrices, use the `Real` option.

New arguments of the `svd` function for computing the “economy size” singular value decomposition

`svd` accepts the new arguments `0` and `'econ'` that let you compute the “economy size” singular value decomposition of a matrix.

`isequaln` function for testing equality of symbolic objects

The MATLAB `isequaln` function tests symbolic objects for equality, treating NaN values as equal.

Control over the order in which `solve` and `vpasolve` functions return solutions

The MATLAB symbolic `solve` and `vpasolve` functions now let you control the order in which they return solutions. To ensure the order of the returned solutions, explicitly specify the independent (input) variables. For example, the syntax `[b, a] = solve(eqns, b, a)` guarantees the order of the returned solutions, while the syntax `[b, a] = solve(eqns)` does not.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
simple	Still runs	simplify	Replace all instances of <code>simple(S)</code> with <code>simplify(S)</code> . There is no replacement for <code>[r, how] = simple(S)</code> .
emlBlock	Warns	matlabFunctionBlock	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .
MuPAD factoring functions <code>numlib::mpqs</code> , <code>numlib::pollard</code> , and <code>numlib::ecm</code>	Warns	<code>ifactor</code>	Replace all instances of <code>numlib::mpqs</code> , <code>numlib::pollard</code> , and <code>numlib::ecm</code> with <code>ifactor</code> .
MuPAD <code>Dom::SparseMatrixF2</code> domain	Warns	<code>Dom::Matrix(Dom::IntegerMod(2))</code>	Replace all instances of <code>Dom::SparseMatrixF2</code> with <code>Dom::Matrix(Dom::IntegerMod(2))</code> .
MuPAD <code>userinfo</code>	Warns	<code>print</code>	Use <code>print</code> instead of <code>userinfo</code> .
MuPAD <code>setuserinfo</code>	Warns	<code>prog::trace</code> or <code>debug</code>	Try using <code>prog::trace</code> or <code>debug</code> instead of <code>setuserinfo</code> .
<code>poly</code>	Errors	<code>charpoly</code>	Replace all instances of <code>poly</code> with <code>charpoly</code> .
<code>sqrt</code> target of the MuPAD <code>simplify</code> function	Errors	MuPAD <code>radsimp</code> or <code>simplifyRadical</code>	Replace all instances of <code>simplify</code> function calls involving the <code>sqrt</code> target with <code>radsimp</code> or <code>simplifyRadical</code> . Alternatively, replace these calls with <code>simplify</code> function calls without targets.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
cos, sin, exp, and ln targets of the MuPAD simplify function	Errors	MuPAD simplify without targets	Replace all instances of simplify function calls involving these targets with simplify function calls without targets. This can lead to a better simplification for some expressions.
MuPAD transform::fourier	Errors	MuPAD fourier	Replace all instances of transform::fourier with fourier.
MuPAD transform::invfourier	Errors	MuPAD ifourier	Replace all instances of transform::invfourier with ifourier.
MuPAD transform::laplace	Errors	MuPAD laplace	Replace all instances of transform::laplace with laplace.
MuPAD transform::invlaplace	Errors	MuPAD ilaplace	Replace all instances of transform::invlaplace with ilaplace.
MuPAD transform::ztrans	Errors	MuPAD ztrans	Replace all instances of transform::ztrans with ztrans.
MuPAD transform::invztrans	Errors	MuPAD iztrans	Replace all instances of transform::invztrans with iztrans.
MuPAD transform::fourier::addpattern	Errors	MuPAD fourier::addpattern	Replace all instances of transform::fourier::addpattern with fourier::addpattern.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD <code>transform::invfourier::addpattern</code>	Errors	MuPAD <code>ifourier::addpattern</code>	Replace all instances of <code>transform::invfourier::addpattern</code> with <code>ifourier::addpattern</code> .
MuPAD <code>transform::laplace::addpattern</code>	Errors	MuPAD <code>laplace::addpattern</code>	Replace all instances of <code>transform::laplace::addpattern</code> with <code>laplace::addpattern</code> .
MuPAD <code>transform::invlaplace::addpattern</code>	Errors	MuPAD <code>ilaplace::addpattern</code>	Replace all instances of <code>transform::invlaplace::addpattern</code> with <code>ilaplace::addpattern</code> .
MuPAD <code>transform::ztrans::addpattern</code>	Errors	MuPAD <code>ztrans::addpattern</code>	Replace all instances of <code>transform::ztrans::addpattern</code> with <code>ztrans::addpattern</code> .
MuPAD <code>transform::invztrans::addpattern</code>	Errors	MuPAD <code>iztrans::addpattern</code>	Replace all instances of <code>transform::invztrans::addpattern</code> with <code>iztrans::addpattern</code> .
MuPAD <code>prog::calledFrom</code>	Errors	<code>context(hold(procname))</code>	Replace all instances of <code>prog::calledFrom()</code> with <code>context(hold(procname))</code> .
MuPAD <code>prog::calltree</code>	Errors	<code>prog::trace</code>	Use <code>prog::trace</code> instead of <code>prog::calltree</code> .
MuPAD <code>prog::error</code>	Errors	<code>getlasterror</code>	Use <code>getlasterror</code> instead of <code>prog::error</code> .
MuPAD <code>prog::memuse</code>	Errors	<code>prog::trace(Mem)</code> or <code>bytes()</code>	Use <code>prog::trace(Mem)</code> or <code>bytes()</code> instead of <code>prog::memuse</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD <code>prog::testfunc</code>	Errors	<code>print(Unquoted, "...")</code>	Use <code>print(Unquoted, "...")</code> instead of <code>prog::testfunc</code> .
MuPAD <code>prog::testmethod</code>	Errors	<code>prog::test(..., Method = myTestMethod)</code>	Use <code>prog::test(..., Method = myTestMethod)</code> instead of <code>prog::testmethod</code> .
MuPAD <code>prog::testnum</code>	Errors	Nothing	No replacement
Dynamic modules for MuPAD, including the <code>module</code> , <code>external</code> , and <code>Pref::unloadableModules</code> functions and all functions of the <code>module</code> library	Errors	Nothing	No replacement

R2012b

Version: 5.9

New Features

Bug Fixes

Compatibility Considerations

MATLAB symbolic matrix analysis functions for characteristic (`charpoly`) and minimal (`minpoly`) polynomials and for norm (`norm`) and condition (`cond`) number

`charpoly` computes the characteristic polynomial of a matrix.

`minpoly` computes the minimal polynomial of a matrix.

`norm` computes the 2-norm (default), 1-norm, Frobenius norm, and infinity norm of a symbolic matrix. It also computes the P-norm, Frobenius norm, infinity norm, and negative infinity norm of a symbolic vector.

`cond` computes the corresponding condition numbers of a matrix.

poles function for determining the poles of an expression

The MATLAB `poles` function determines the poles of a symbolic expression or function. The `poles` function is also implemented in MuPAD.

`vpasolve` function for solving equations and systems using variable precision arithmetic

The MATLAB `vpasolve` function solves equations and systems of equations numerically.

Functions for converting linear systems of equations to matrix form $AX=B$ (`equationsToMatrix`) and solving matrix equations (`linsolve`)

The MATLAB `equationsToMatrix` function converts a linear system of equations to the matrix form $AX = B$. The function returns the coefficient matrix A and the vector B that contains the right sides of the equations.

The MATLAB `linsolve` function solves linear systems of equations represented in the matrix form $AX = B$. The function also returns the reciprocal of the condition number of the square coefficient matrix A . If A is rectangular, `linsolve` returns the rank of A .

MATLAB symbolic functions for describing pulses: `rectangularPulse` and `triangularPulse`

`rectangularPulse` and `triangularPulse` compute the rectangular and triangular pulse functions, respectively.

In MuPAD, the new `rectangularPulse` and `triangularPulse` functions are equivalent to `rectpulse` and `tripulse`, respectively.

MuPAD functions for computing integral and Z-transforms

These new MuPAD functions compute integral and Z-transforms:

- `fourier` computes the Fourier transform. You can specify the parameters of the Fourier transform using the new `Pref::fourierParameters` function.
- `ifourier` computes the inverse Fourier transform. You can specify the parameters of the inverse Fourier transform using the new `Pref::fourierParameters` function.
- `laplace` computes the Laplace transform.
- `ilaplace` computes the inverse Laplace transform.
- `ztrans` computes the Z-transform.
- `iztrans` computes the inverse Z-transform.

MuPAD `Pref::fourierParameters` function for specifying Fourier parameters

The MuPAD `Pref::fourierParameters` function lets you specify parameters for Fourier and inverse Fourier transforms.

MuPAD functions for adding transform patterns

These new MuPAD functions add new patterns for integral and Z-transforms:

- `fourier::addpattern` adds new patterns for the Fourier transform.
- `ifourier::addpattern` adds new patterns for the inverse Fourier transform.
- `laplace::addpattern` adds new patterns for the Laplace transform.

- `ilaplace::addpattern` adds new patterns for the inverse Laplace transform.
- `ztrans::addpattern` adds new patterns for the Z-transform.
- `iztrans::addpattern` adds new patterns for the inverse Z-transform.

MuPAD does not save custom patterns permanently. The new patterns are available in the *current* MuPAD session only.

noFlatten option of the MuPAD proc function for preventing sequence flattening

The MuPAD `proc` function accepts the new `noFlatten` option. This option prevents flattening of sequences passed as arguments of the procedure.

testtype uses testtypeDom slot for overloading by the second argument

If in the call `testtype(object, T)` the argument `T` is a domain, then the method `testtypeDom` of `T` is called with the arguments `object, T`. If `T` is not a domain, then the method `testtypeDom` of `T::dom` is called with the arguments `object, T`.

Compatibility Considerations

In previous releases, `testtype` used the `testtype` slot for overloading by the second argument.

New upper limit on the number of digits in double

By default, the working precision for `double` is now limited to at most by 664 digits. You can explicitly specify a larger precision using `digits`.

Compatibility Considerations

Some results returned by `double` can differ from previous releases. For example, in previous releases `double` approximated the expression

```
x = sym('400!*((exp(2000)+1)/(exp(2000) - 1) - 1)')
```

by 3.2997. Now it approximates this expression by 0.

To get the same result as in previous releases, increase the precision of computations:

```
digits(1000)
double(x)

ans =
  3.2997
```

New definition for real and imag

Starting in R2012a, `real` and `imag` are no longer defined via `conj`. They use the MuPAD `Re` and `Im` functions instead.

Compatibility Considerations

In R2011b and earlier, `real` and `imag` are defined via the `conj` function:

```
syms z
real(z)
imag(z)

ans =
z/2 + conj(z)/2

ans =
- (z*i)/2 + (conj(z)*i)/2
```

Therefore, `real` and `imag` can return results in a different form. Results returned by `real` and `imag` now are mathematically equivalent to the results returned in previous releases.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Old syntax of <code>taylor</code>	Errors	New calling syntax	Update all instances of <code>taylor</code> function calls using the new syntax.
<code>char(A, d)</code>	Errors	<code>char(A)</code>	Replace all instances of <code>char(A, d)</code> with <code>char(A)</code> .
<code>poly</code>	Warns	<code>charpoly</code>	Replace all instances of <code>poly</code> with <code>charpoly</code> .
<code>emlBlock</code>	Warns	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .
Ability to create links from MuPAD notebooks to MuPAD documentation pages	Not available	Nothing	No replacement
<code>openmuphlp</code>	Errors	Nothing	No replacement
MuPAD Help Browser	Not available	Documentation Center	MuPAD documentation is now available in Documentation Center.
MuPAD Editor	Not available	MATLAB Editor	Open and edit MuPAD program files (<code>.mu</code> files) in the MATLAB Editor. The MATLAB Editor supports syntax highlighting and smart indenting for these files.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>psi(k0:k1,X)</code>	Errors	<code>psi(k,X)</code> , where <code>k</code> is a scalar specifying the <code>k</code> th derivative of <code>psi</code> at the elements of <code>X</code> .	<p>Replace all instances of <code>psi(k0:k1,X)</code> with <code>psi(k,X)</code>, where <code>k</code> is a scalar. To modify your code, loop through the values <code>k0:k1</code>. For example:</p> <pre>for k = k0:k1 Y(:,k) = psi(k,X); end</pre> <p>In a future release, <code>size(Y)</code> will be <code>size(X)</code>. Modify any code that depends on <code>size(Y)</code>.</p>
<code>diff</code> and <code>int</code> methods for inputs of the <code>char</code> type	Errors	<code>sym</code>	Use the <code>sym</code> method instead.
<code>sqrt</code> target of the MuPAD <code>simplify</code> function	Errors	MuPAD <code>radsimp</code> or <code>simplifyRadical</code>	Replace all instances of <code>simplify</code> function calls involving the <code>sqrt</code> target with <code>radsimp</code> or <code>simplifyRadical</code> . Alternatively, replace these calls with <code>simplify</code> function calls without targets.
<code>cos</code> , <code>sin</code> , <code>exp</code> , and <code>ln</code> targets of the MuPAD <code>simplify</code> function	Errors	MuPAD <code>simplify</code> without targets	Replace all instances of <code>simplify</code> function calls involving these targets with <code>simplify</code> function calls without targets. This can lead to a better simplification for some expressions.
MuPAD <code>transform::fourier</code>	Warns	MuPAD <code>fourier</code>	Replace all instances of <code>transform::fourier</code> with <code>fourier</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD <code>transform::invfourier</code>	Warns	MuPAD <code>ifourier</code>	Replace all instances of <code>transform::invfourier</code> with <code>ifourier</code> .
MuPAD <code>transform::laplace</code>	Warns	MuPAD <code>laplace</code>	Replace all instances of <code>transform::laplace</code> with <code>laplace</code> .
MuPAD <code>transform::invlaplace</code>	Warns	MuPAD <code>ilaplace</code>	Replace all instances of <code>transform::invlaplace</code> with <code>ilaplace</code> .
MuPAD <code>transform::ztrans</code>	Warns	MuPAD <code>ztrans</code>	Replace all instances of <code>transform::ztrans</code> with <code>ztrans</code> .
MuPAD <code>transform::invztrans</code>	Warns	MuPAD <code>iztrans</code>	Replace all instances of <code>transform::invztrans</code> with <code>iztrans</code> .
MuPAD <code>transform::fourier::addpattern</code>	Warns	MuPAD <code>fourier::addpattern</code>	Replace all instances of <code>transform::fourier::addpattern</code> with <code>fourier::addpattern</code> .
MuPAD <code>transform::invfourier::addpattern</code>	Warns	MuPAD <code>ifourier::addpattern</code>	Replace all instances of <code>transform::invfourier::addpattern</code> with <code>ifourier::addpattern</code> .
MuPAD <code>transform::laplace::addpattern</code>	Warns	MuPAD <code>laplace::addpattern</code>	Replace all instances of <code>transform::laplace::addpattern</code> with <code>laplace::addpattern</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD <code>transform::invlaplace::addpattern</code>	Warns	MuPAD <code>ilaplace::addpattern</code>	Replace all instances of <code>transform::invlaplace::addpattern</code> with <code>ilaplace::addpattern</code> .
MuPAD <code>transform::ztrans::addpattern</code>	Warns	MuPAD <code>ztrans::addpattern</code>	Replace all instances of <code>transform::ztrans::addpattern</code> with <code>ztrans::addpattern</code> .
MuPAD <code>transform::invztrans::addpattern</code>	Warns	MuPAD <code>iztrans::addpattern</code>	Replace all instances of <code>transform::invztrans::addpattern</code> with <code>iztrans::addpattern</code> .
MuPAD <code>prog::calledFrom</code>	Warns	<code>context(hold(procname))</code>	Replace all instances of <code>prog::calledFrom()</code> with <code>context(hold(procname))</code> .
MuPAD <code>prog::calltree</code>	Warns	<code>prog::trace</code>	Use <code>prog::trace</code> instead of <code>prog::calltree</code> .
MuPAD <code>prog::error</code>	Warns	<code>getlasterror</code>	Use <code>getlasterror</code> instead of <code>prog::error</code> .
MuPAD <code>prog::memuse</code>	Warns	<code>prog::trace(Mem)</code> or <code>bytes()</code>	Use <code>prog::trace(Mem)</code> or <code>bytes()</code> instead of <code>prog::memuse</code> .
MuPAD <code>prog::testfunc</code>	Warns	<code>print(Unquoted, "...")</code>	Use <code>print(Unquoted, "...")</code> instead of <code>prog::testfunc</code> .
MuPAD <code>prog::testmethod</code>	Warns	<code>prog::test(..., Method = myTestMethod)</code>	Use <code>prog::test(..., Method = myTestMethod)</code> instead of <code>prog::testmethod</code> .
MuPAD <code>prog::testnum</code>	Warns	Nothing	No replacement

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Dynamic modules for MuPAD, including the <code>module</code> , <code>external</code> , and <code>Pref::unloadableModules</code> functions and all functions of the <code>module</code> library	Warns	Nothing	No replacement

R2012a

Version: 5.8

New Features

Bug Fixes

Compatibility Considerations

New Special Functions

The following special functions are available:

- `airy` computes the Airy functions of the first and the second kinds. It also computes the first derivatives of the Airy functions.
- `beta` computes the beta function.
- `erfinv` and `erfcinv` compute the inverse and inverse complementary error functions.
- `factorial` computes the factorial function.
- `nchoosek` computes binomial coefficients.
- `whittakerM` and `whittakerW` compute the Whittaker M and Whittaker W functions.

New Vector Analysis Functions

The following vector analysis functions are available:

- `curl` computes the curl of a vector field.
- `divergence` computes the divergence of a vector field.
- `laplacian` computes the laplacian of a scalar function.
- `potential` computes the scalar potential of a vector field.
- `vectorPotential` computes the vector potential of a three-dimensional vector field.

Computations with Symbolic Functions

The toolbox lets you create symbolic functions. For details, see [Creating Symbolic Functions](#).

`dsolve`, `ezplot`, the new `odeToVectorField` function, and other Symbolic Math Toolbox functions now support computations with symbolic functions.

The toolbox also provides the following functions to support common operations on symbolic functions:

- `argnames` returns a symbolic array of all input variables of a symbolic function.
- `formula` returns a mathematical expression that defines the symbolic function.

Assumptions on Variables

You can set assumptions on symbolic variables by using these functions:

- `assume` sets assumptions on symbolic variables.
- `assumeAlso` adds assumptions on symbolic variables without erasing the previous assumptions.
- `assumptions` shows assumptions set on symbolic variables.

New Relational Operators Create Equations, Inequalities, and Relations

Use these relational operators to create symbolic equations, inequalities, and relations:

- `==` and its functional form `eq` create a symbolic equation. You can solve these equations using `solve` or `dsolve`, plot them using `ezplot`, set assumptions using `assume` or `assumeAlso`, or use them in logical expressions.
- `~=` and its functional form `ne` create a symbolic inequality. You can use inequalities in assumptions and logical expressions.
- `>`, `>=`, `<`, `<=`, and their functional forms `ge`, `gt`, `le`, and `lt` create symbolic relations. You can use relations in assumptions and logical expressions.

Compatibility Considerations

In previous releases, `eq` evaluated equations and returned logical 1 or 0. Now it returns unevaluated equations letting you create equations that you can pass to `solve`, `assume`, and other functions. To obtain the same results as in previous releases, wrap equations in `logical` or `isAlways`. For example, use `logical(A == B)`.

New Logical Operators Create Logical Expressions

Use these logical operations let you create logical expressions with symbolic subexpressions:

- `&` or its functional form `and` defines the logical conjunction (the logical AND) for symbolic expressions.
- `|` or its functional form `or` defines the logical disjunction (the logical OR) for symbolic expressions.

- `~` or its functional form `not` defines the logical negation (the logical NOT) for symbolic expressions.
- `xor` defines the logical exclusive disjunction (the logical XOR) for symbolic expressions.

If logical expressions are elements of a symbolic array, you can use these new functions to test the logical expressions:

- `all` tests whether all equations and inequalities represented as elements of a symbolic array are valid.
- `any` tests whether at least one of equations and inequalities represented as elements of a symbolic array is valid.

New Functions Test Validity of Symbolic Equations, Inequalities, and Relations

Use these functions to test symbolic equations, inequalities, and relations, including logical statements:

- `isAlways` checks whether an equation, inequality, or relation holds for all values of its variables.
- `logical` checks the validity of an equation, inequality, or relation. This function does not simplify or mathematically transform expressions that form an equation, inequality, or relation. It also typically ignores assumptions on variables.

New Functions Manipulate Symbolic Expressions

These functions provide more flexible options for manipulating symbolic expressions:

- The `rewrite` function rewrites expressions in terms of target functions. It returns a mathematically equivalent form of an expression using the specified target functions. For example, it can rewrite trigonometric expressions using the exponential function.
- `children` returns child subexpressions, or terms, of a symbolic expression.

New `odeToVectorField` Function Converts Higher-Order Differential Equations to Systems of First-Order Differential Equations

`odeToVectorField` converts second- and higher-order differential equations to systems of first-order differential equations. It returns a symbolic vector representing the

resulting system of first-order differential equations. With `matlabFunction` you can generate a MATLAB function from this vector, and then use it as an input for the MATLAB numerical solvers `ode23` and `ode45`.

In MuPAD, the new `numeric::odeToVectorField` function is equivalent to `numeric::ode2vectorfield`.

New Calling Syntax for the `taylor` Function

The `taylor` function that computes the Taylor series expansions has a new syntax and set of options.

Compatibility Considerations

The new syntax is not valid before Version 5.8. The old syntax is still supported, but will be removed in a future release. To update existing code that relies on the old syntax, make the following changes to the `taylor` function calls:

- Specify the truncation order using the name-value pair argument `Order`.
- Specify the expansion point using the name-value pair argument `ExpansionPoint`.

Alternatively, specify the expansion point as a third input argument. In this case, you must also specify the independent variable or the vector of variables as the second input argument.

For details and examples, see `taylor`.

New MuPAD Functions Compute Rectangular and Triangular Pulse Functions

The MuPAD `rectpulse` and `tripulse` functions compute the rectangular and triangular pulse functions, respectively.

MuPAD `det`, `linalg::det`, `inverse`, `linsolve`, and `linalg::matlinsolve` Functions Accept the New `Normal` Option

The MuPAD `det`, `linalg::det`, `inverse`, `linsolve`, and `linalg::matlinsolve` functions accept the new `Normal` option that guarantees normalization of the returned

results. The `_invert` methods of the MuPAD `Dom::Matrix(R)` and `Dom::DenseMatrix(R)` domains also accept `Normal`.

MuPAD `linalg::matlinsolve` Function Accepts the New `ShowAssumptions` Option

The MuPAD `linalg::matlinsolve` function accepts the new `ShowAssumptions` option. This option lets you see internal assumptions on symbolic parameters that `linalg::matlinsolve` makes while solving a system of equations.

Enhanced MuPAD `pdivide` Function

Enhanced MuPAD `pdivide` function now performs pseudo-division of multivariate polynomials.

Improved MuPAD `prog::remember` Function

Improved MuPAD `prog::remember` function, which lets you use the `remember` mechanism in procedures streamlines such processes as debugging, profiling, and argument checking.

Functionality Being Removed or Changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Old syntax of <code>taylor</code>	Warns	New syntax	Update all instances of <code>taylor</code> function calls using the new syntax.
Default number of simplification steps in <code>simplify</code> has changed from 50 to 100.	Uses the new default setting	<code>simplify(S, 'Steps', 50)</code>	To terminate algebraic simplification after 50 steps, call <code>simplify</code> with the name-value pair argument <code>'Steps', 50</code> .

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>char(A,d)</code>	Warns	<code>char(A)</code>	Replace all instances of <code>char(A,d)</code> with <code>char(A)</code> .
<code>emlBlock</code>	Warns	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .
<code>psi(k0:k1,X)</code>	Warns	<code>psi(k,X)</code> where <code>k</code> is a scalar specifying the <code>k</code> th derivative of <code>psi</code> at the elements of <code>X</code> .	<p>Replace all instances of <code>psi(k0:k1,X)</code> with <code>psi(k,X)</code>, where <code>k</code> is a scalar. To modify your code, loop through the values <code>k0:k1</code>. For example:</p> <pre>for k = k0:k1 Y(:,k) = psi(k,X); end</pre> <p>In the future, <code>size(Y)</code> will be <code>size(X)</code>. Modify any code that depends on <code>size(Y)</code>.</p>
<code>sqrt</code> target of the MuPAD <code>simplify</code> function	Warns	MuPAD <code>radsimp</code> or <code>simplifyRadical</code>	Replace all instances of <code>simplify</code> function calls involving the <code>sqrt</code> target with <code>radsimp</code> or <code>simplifyRadical</code> . Alternatively, replace these calls with <code>simplify</code> function calls without targets.
<code>cos</code> , <code>sin</code> , <code>exp</code> , and <code>ln</code> targets of the MuPAD <code>simplify</code> function	Warns	MuPAD <code>simplify</code> without targets	Replace all instances of <code>simplify</code> function calls involving these targets with <code>simplify</code> function calls without targets. This can lead to a better simplification for some expressions.
MuPAD frame function	Errors	Nothing	No replacement

R2011b

Version: 5.7

New Features

Bug Fixes

Compatibility Considerations

MATLAB Editor Now Supports MuPAD Program Files

You can open and edit MuPAD program files (.mu files) in the MATLAB Editor. MATLAB Editor supports syntax highlighting and smart indenting for these files.

dsolve, expand, int, simple, simplify, and solve Accept More Options

`dsolve` now accepts the `IgnoreAnalyticConstraints` and `MaxDegree` options.

`expand` now accepts the `ArithmeticOnly` and `IgnoreAnalyticConstraints` options.

`int` now accepts the `IgnoreAnalyticConstraints`, `IgnoreSpecialCases`, and `PrincipalValue` options.

`simple` now accepts the `IgnoreAnalyticConstraints` option.

`simplify` now accepts the `IgnoreAnalyticConstraints`, `Seconds`, and `Steps` options.

`solve` now accepts the `IgnoreAnalyticConstraints`, `IgnoreProperties`, `MaxDegree`, `PrincipalValue`, and `Real` options.

New read Function Reads MuPAD Program Files in MATLAB

`read` simplifies using your own MuPAD procedures in MATLAB. See [Before Calling a Procedure](#) for details.

New symprod Function Computes Products of Series

`symprod` computes definite and indefinite products of symbolic series.

New hessian Function Computes Hessian Matrices

`hessian` computes the Hessian matrix of a scalar function.

New gradient Function Computes Vector Gradients

`gradient` computes the vector gradient of a scalar function in Cartesian coordinates. In MuPAD, the new `linalg::gradient` function is equivalent to `linalg::grad`.

New erfc Function Computes the Complementary Error Function

`erfc` computes the complementary error function.

New psi Function Computes the Digamma and Polygamma Functions

`psi` computes the digamma and polygamma functions.

New wrightOmega Function Computes the Wright omega Function

`wrightOmega` computes the Wright omega function.

New simplifyFraction Function Simplifies Expressions

`simplifyFraction` returns a simplified form of a fraction where both numerator and denominator are polynomials and their greatest common divisor is 1. In MuPAD, the new `simplifyFraction` function is equivalent to `normal`.

New MuPAD simplifyRadical Function Simplifies Radicals in Arithmetical Expressions

The new MuPAD `simplifyRadical` function is equivalent to the MuPAD `radsimp` function.

pretty Function Now Uses Abbreviations in Long Output Expressions for Better Readability

`pretty` uses abbreviations when presenting symbolic results in the MATLAB Command Window. This new format of presenting symbolic results enhances readability of long output expressions.

MuPAD normal Function Accepts the New Expand Option

The MuPAD `normal` function accepts the new `Expand` option that determines whether numerators and denominators of fractions are expanded.

Compatibility Considerations

In previous releases, `normal` returned a fraction with the expanded numerator and denominator by default. Now the default setting is that `normal` can return factored expressions in numerators and denominators. In explicit calls to `normal`, you can use the `Expand` option to get the same behavior as in previous releases.

If a function calls `normal` internally, then that function can return its results in a different form. These new results are mathematically equivalent to the results that you get in previous releases. Many MuPAD library functions can call `normal`.

Modified MuPAD groebner Library

All functions of the MuPAD `groebner` library now can accept and return polynomials with arbitrary arithmetical expressions.

MuPAD groebner::gbasis Function Accepts the New Factor and IgnoreSpecialCases Options

With the `Factor` option, `groebner::gbasis` returns a set of lists, such that:

- Each list is the Groebner basis of an ideal.
- The union of these ideals is a superset of the ideal given as input, and a subset of the radical of that ideal.

With the `IgnoreSpecialCases` option, `groebner::gbasis` handles all coefficients in all intermediate results as nonzero unless these coefficients are equal to 0 for all parameter values.

New MuPAD Functions for Computing Logarithms

The new MuPAD `log2` and `log10` functions compute logarithms to the bases 2 and 10, respectively. Also, in MuPAD `log(x)` is now an alias for `ln(x)`.

Functionality Being Removed or Changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>emlBlock</code>	Warns	<code>matlabFunctionBlock</code>	Replace all instances of <code>emlBlock</code> with <code>matlabFunctionBlock</code> .
Real and <code>IgnoreProperties</code> options in MuPAD <code>ode::solve</code>	Warns	<code>IgnoreSpecialCases</code> or <code>IgnoreAnalyticConstraints</code>	Try using <code>IgnoreSpecialCases</code> or <code>IgnoreAnalyticConstraints</code> instead.

R2011a

Version: 5.6

New Features

Bug Fixes

Compatibility Considerations

Expression Wrapping of Math Output in the MuPAD Notebook Interface

The new default format of presenting results enhances readability by wrapping long output expressions, including long numbers, fractions and matrices.

Symbolic Solver Handles More Non-Algebraic Equations

The enhanced `rationalize` function in MuPAD helps the symbolic solver to handle more systems of non-algebraic equations. In particular, this improvement enables the toolbox to solve more systems of trigonometric equations.

Improved Performance in the Ordinary Differential Equation Solver

The ordinary differential equation solver demonstrates better performance.

Improved Performance for Polynomial Arithmetic Operations

The MuPAD functions `gcdex`, `partfrac`, `polylib::resultant`, and `solvelib::pdioe` now demonstrate better performance.

New MuPAD `polylib::subresultant` Function Computes Subresultants of Polynomials

`polylib::subresultant` computes subresultants of two polynomials or polynomial expressions.

MuPAD `partfrac` Function Accepts the New List Option

With the new `List` option, `partfrac` returns a list consisting of the numerators and denominators of the partial fraction decomposition.

New MuPAD `inverf` and `inverfc` Functions Compute the Inverses of Error Functions

The `inverf` function computes the inverse of the error function.

The `inverfc` function computes the inverse of the complementary error function.

New MuPAD `numlib::checkPrimalityCertificate` Function Tests Primality Certificates

`numlib::checkPrimalityCertificate` tests primality certificates returned by `numlib::proveprime`. For information about proving primality of numbers, see “Proving Primality” in the MuPAD documentation.

New Demos

There are three new demos that show how to solve equations and compute derivatives and integrals:

- Solving Algebraic and Differential Equations
- Differentiation
- Integration

To run the new demos, enter `symeqndemo`, `syndiffdemo`, or `symintdemo` in the MATLAB Command Window.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
MuPAD <code>matchlib::analyze</code>	Errors	MuPAD <code>prog::exprtree</code>	To visualize expressions, use <code>prog::exprtree</code> .
MuPAD <code>prog::testcall</code>	Errors	Nothing	No replacement
MuPAD <code>prog::testerrors</code>	Errors	Nothing	No replacement

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Old syntax of MuPAD <code>prog::getOptions</code>	Errors	The new syntax	Update all instances of <code>prog::getOptions</code> calls using the new syntax.
Old syntax of MuPAD <code>prog::trace</code>	Errors	The new syntax	Update all instances of <code>prog::trace</code> calls using the new syntax.

R2010b

Version: 5.5

New Features

Bug Fixes

Compatibility Considerations

sym Function Creates Matrices of Symbolic Variables

The `sym` function now provides a shortcut for creating vectors and matrices of symbolic variables.

For more information, see [Creating a Matrix of Symbolic Variables](#).

generate::Simscape Function Generates Simscape Equations from MuPAD Expressions

The new MuPAD function `generate::Simscape` converts MuPAD expressions to Simscape equations.

MuPAD Code Generation Functions Accept the New NoWarning Option

MuPAD functions `generate::C`, `generate::fortran`, `generate::MATLAB`, and `generate::Simscape` accept the new `NoWarning` option. The option suppresses all warnings issued by these functions.

Improved MuPAD Hyperlink Dialog Box

Creating and editing links in MuPAD has become easier with the improved Hyperlink dialog box.

MuPAD Notebook Highlights Matched and Unmatched Delimiters

MuPAD Notebook now can notify you about matched and unmatched delimiters such as parentheses, brackets, and braces.

Improved Performance When Solving Linear Systems in a Matrix Form

MuPAD `linalg::matlinsolve` function, which solves linear systems of equations in a matrix form, demonstrates better performance.

MuPAD Solver for Ordinary Differential Equations Handles More Equation Types

Enhanced MuPAD solver handles more first-order nonlinear and third-order linear ordinary differential equations. The solver demonstrates improved performance.

New Syntax for the MuPAD `prog::getOptions` Function

The `prog::getOptions` function that collects and verifies options within a procedure has the new syntax.

Compatibility Considerations

The new syntax is not valid in MuPAD versions earlier than 5.5. The old syntax is supported in MuPAD 5.5, but will be removed in a future release.

New Syntax for the MuPAD `prog::trace` Function

The `prog::trace` function used for debugging has the new syntax. The function observes entering and exiting the MuPAD functions.

Compatibility Considerations

The new syntax is not valid in MuPAD versions earlier than 5.5. The old syntax is not supported in MuPAD 5.5.

Improved Interface for Arithmetical Operations on Polynomials

Improved interface for arithmetical operations between polynomials and arithmetical expressions. In previous releases, to perform an arithmetical operation on a polynomial and an arithmetical expression, you must explicitly convert that expression to a polynomial of the corresponding type. Now, when you operate on a polynomial and an arithmetical expression, MuPAD internally converts the arithmetical expression to a polynomial and performs the calculation.

MuPAD `igcd` Function Now Accepts Complex Numbers as Arguments

The MuPAD `igcd` function, which computes the greatest common divisor of integers, now accepts complex numbers. Both real and imaginary parts of accepted complex numbers must be integers or arithmetic expressions that represent integers.

Enhanced Solver For Factorable Polynomial Systems

The MuPAD `solve` function performs better on factorable polynomial systems.

MuPAD Now Evaluates Large Sums with Subtractions Faster

MuPAD performs evaluations of large sums that contain subtractions faster than in previous releases.

Compatibility Considerations

In MuPAD, the difference operator (`-`) no longer invokes the `_subtract` function. Instead, it invokes the `_plus` and `_negate` functions. For example, `a - b` is equivalent to `_plus(a, _negate(b))`.

MuPAD `freeIndets` Function Accepts the New `All` Option

The `freeIndets` function accepts the new `All` option. With this option, `freeIndets` does not exclude the 0th operand from the list of free identifiers.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>diff</code> and <code>int</code> methods for inputs of the <code>char</code> type	Warns	<code>sym</code>	Use the <code>sym</code> method instead.
MuPAD <code>matchlib::analyze</code>	Warns	MuPAD <code>prog::expree</code>	To visualize expressions, use <code>prog::expree</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
MuPAD <code>prog::testcall</code>	Warns	None	No replacement
MuPAD <code>prog::testerrors</code>	Warns	None	No replacement
The following options in MuPAD <code>prog::trace:</code> <ul style="list-style-type: none"> • All • Backup • Force • Name • Proc • Plain • Width 	Errors	None	No replacement. These options are not supported in the current release.
Global properties in MuPAD	Errors	Assumptions on each variable	Make assumptions on each variable instead.

R2010a

Version: 5.4

New Features

Bug Fixes

Compatibility Considerations

When Opening Notebook, MuPAD Can Jump to Particular Locations

The `mupad` command that opens a MuPAD notebook now supports references to particular places inside a notebook. You can create a link target inside a notebook and refer to it when opening a notebook.

`simscapeEquation` Function Generates Simscape Equations from Symbolic Expressions

The new `simscapeEquation` command represents symbolic expressions in the form of Simscape equations. For more information, see *Generating Simscape Equations in the Symbolic Math Toolbox* documentation.

New Calling Syntax for the `sort` Function

The `sort` function that sorts the element of symbolic arrays and polynomials has the new syntax and set of options.

Compatibility Considerations

In previous releases, the `sort` function flattened symbolic matrices to vectors before sorting the elements. Now the `sort` function sorts the elements of each column or each row of a symbolic matrix. If you want to obtain the same results as in the previous release, flatten the symbolic matrix before sorting it: `sort(A(:))`.

Changes in the `symengine` Function

The toolbox no longer supports the ability to choose an alternative symbolic engine.

64-Bit GUI Support for Macintosh

MuPAD now supports 64-bit graphical user interfaces (such as notebooks and Editor and Debugger windows) for a 64-bit Macintosh operating system.

New MuPAD Print Preview Dialog

Adjusting MuPAD documents for printing is easier with the new Print Preview dialog. You can view one or several pages, zoom in and out, switch between page orientations,

adjust the page settings without closing the dialog, and print the page or save it to PDF format

Improved Configure MuPAD Dialog Box

Specifying the default settings for graphical user interfaces, such as notebooks and Editor and Debugger windows, has become easier with the improved configuration dialog box.

MuPAD Support for Basic Arithmetic Operations for Lists

Basic arithmetic operations now work for lists.

Improved Performance When Operating on Matrices with Symbolic Elements

MuPAD demonstrates better performance when handling some linear algebra operations on matrices containing symbolic elements.

Enhanced MuPAD `divide` Function

Enhanced MuPAD `divide` function computes the quotient and remainder for division of multivariate polynomials.

Improved Performance for Operations on Polynomials

Improved performance for conversions involving polynomials. Improved performance for operations on polynomials including evaluation, multiplication, and division.

Compatibility Considerations

If the coefficients of a polynomial contain the variables of the polynomial itself, the form of results returned by the MuPAD `poly` function can differ from previous releases. In previous releases, the `poly` function converted such coefficients to monomials. Now the `poly` function can return the coefficients of the original expression as coefficients in the resulting polynomial. To get the same behavior as in previous releases, use `expr` to

convert an original polynomial into an expression, and then call the `poly` function. For example, the following call exercises the old behavior: `poly(expr(p), [y, x])`.

MuPAD `coeff` Function Accepts the New `All` Option

The `coeff` function accepts the `newAll` option. With this option, `coeff` returns all coefficients of a polynomial including those equal to 0.

MuPAD `expand` Function Accepts the New `ArithmeticOnly` Option

The `expand` function accepts the new `ArithmeticOnly` option. The option allows you to expand a sum without expanding trigonometric expressions and special functions in its terms. Technically, the option omits overloading the `expand` function for each term of the original expression.

MuPAD `expand` Function Now Expands Powers of Products

The `expand` function now expands powers of products such as $(xy)^n$ for positive x and y . When called with the `IgnoreAnalyticConstraints` option, the function expands the power of products for arbitrary terms.

New Calling Syntax for MuPAD `rationalize` Function

The `rationalize` function that transforms an arbitrary expression into a rational expression has the new syntax and set of options.

Compatibility Considerations

The new syntax is not valid in MuPAD versions earlier than 5.4. The old syntax is supported in MuPAD 5.4, but will be removed in a future release.

Enhanced MuPAD `simplify` and `Simplify` Functions

Enhanced simplification functions, `simplify` and `Simplify`, demonstrate better results for expressions involving trigonometric and hyperbolic functions, square roots, and sums over roots of unity.

MuPAD subs Function Accepts the New EvalChanges Option

The `subs` function now accepts the new `EvalChanges` option. By default, `subs` does not evaluate an expression after making substitutions. With this option, `subs` evaluates all subexpressions that contain substitutions.

MuPAD Solver for Ordinary Differential Equations Handles More Equation Types

Enhanced MuPAD solver handles more second-order linear and first-order nonlinear ordinary differential equations. The solver demonstrates improved performance.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
MuPAD Domain <code>Dom::Ideal</code>	Errors	<code>groebner</code>	Represent ideals as lists, and use functions of the <code>groebner</code> package instead.
MuPAD student library	Errors	<code>plot::Integral</code> and <code>linalg</code>	Use <code>plot::Integral</code> and the <code>linalg</code> package instead.
MuPAD <code>relation</code> option in <code>simplify</code>	Errors	None	No replacement
Global property	Warns	Assumptions on each variable	Make assumptions on each variable instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
digits and vpa do not let you set the number of digits to 1.	Errors	Errors	It is no longer possible to set the number of digits to 1 when using the digits and vpa functions. The Symbolic Math Toolbox software version number 4.9 and lower allowed you to set the number of digits to 1.

R2009b

Version: 5.3

New Features

Bug Fixes

Compatibility Considerations

Support for Windows x64 and 64-Bit Macintosh

The toolbox now supports 64-bit Windows® and Macintosh operating systems. If you work in the MuPAD Notebook Interface on a 64-bit Macintosh operating system, MuPAD runs a 64-bit engine with 32-bit graphical user interfaces, such as notebooks and Editor and Debugger windows.

sym and syms Use Reserved Words as Variable Names

`sym` and `syms` commands now treat reserved MuPAD words, except `pi`, as variable names.

Compatibility Considerations

In previous releases, the reserved words returned MuPAD values. If your code uses the reserved words as MuPAD commands, modify your code and use the `evalin` command with the reserved word as a name. For example, use `evalin(symengine, 'beta')`.

Toolbox Now Displays Floating-Point Results with Their Original Precision

The toolbox now displays the floating-point results with the original precision with which the toolbox returned them.

Compatibility Considerations

In previous releases, the toolbox displayed floating-point results with the current precision. You must update the existing code that relies on the output precision for displaying floating-point numbers. Use `digits` to set the precision you need before computing such results. The toolbox displays the results with the same number of digits it used to compute the results. The toolbox also can increase the specified precision of calculations by several digits.

In previous releases, `sym(A, 'f')` represented numbers in the form $(2^e + N \cdot 2^{(e - 52)})$ or $-(2^e + N \cdot 2^{(e - 52)})$, with integers for N and e , and $N \geq 0$. Now `sym(A, 'f')` displays results in the rational form that actually represents the double-precision floating-point numbers.

New MuPAD Preference `Pref::outputDigits` Controls Floating-Point Outputs

New preference `Pref::outputDigits` controls the precision MuPAD uses to display floating-point results.

Solver for Ordinary Differential Equations Handles More Equation Types

Enhanced solvers handle more equation types of second-order homogeneous linear ordinary differential equations. The solver demonstrates improved performance.

MuPAD limit Function Supports Limits for Incomplete Gamma Function and Exponential Integral Function

Enhanced limit function now can compute limits for incomplete Gamma function and exponential integral function.

Enhanced Simplification Routines for MuPAD Special Functions

Enhanced simplification routines for MuPAD `hypergeom`, `mejerG`, and `bessel` special functions.

Enhanced MuPAD `combine` Function for Logarithms

Enhanced `combine` function demonstrates better performance for logarithms.

MuPAD `normal` Function Accepts New Options

The `normal` command now accepts the options `NoGcd`, `ToCancel`, `Rationalize`, `Recursive`, and `Iterations`. The options control costly operations, such as recognizing greatest common divisors and algebraic dependencies.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
MuPAD Domain <code>Dom::Ideal</code>	Warns	<code>groebner</code>	Represent ideals as lists, and use functions of the <code>groebner</code> package instead.
MuPAD student library	Warns	<code>plot::Integral</code> and <code>linalg</code>	Use <code>plot::Integral</code> and the <code>linalg</code> package instead.
<code>d in char(A, d)</code>	Warns	None	No replacement
MuPAD relation option in <code>simplify</code>	Warns	None	No replacement

R2009a

Version: 5.2

New Features

Bug Fixes

Compatibility Considerations

dsolve Accepts the New Option IgnoreAnalyticConstraints

The `dsolve` command now accepts the option `IgnoreAnalyticConstraints`. The option controls the level of mathematical rigor that the solver uses on the analytical constraints on the solution. By default, the solver ignores all analytical constraints.

Compatibility Considerations

The results of the `dsolve` command can differ from those returned in the previous release. If you want to obtain the same solutions as in the previous release, set the value of the option `IgnoreAnalyticConstraints` to `none`.

emlBlock Function Generates Embedded MATLAB Function Blocks from Symbolic Objects

The new `emlBlock` command converts symbolic expressions to Embedded MATLAB® Function Blocks. You can use these blocks in any Simulink® installation, even those without a Symbolic Math Toolbox license. For more information, see [Generating Embedded MATLAB Blocks in the Symbolic Math Toolbox documentation](#).

matlabFunction Improves Control over Input and Output Parameters

`matlabFunction` now accepts multiple expressions and cell arrays of symbolic arrays as input parameters. The function now allows you to specify the names of the output parameters.

Compatibility Considerations

In previous releases, the default name of an output variable was `RESULT`. Now the default names of the output variables coincide with the names you use to call `matlabFunction`. You must update existing code that relies on the default output name `RESULT`. You can change your code using any of these methods:

- Define the name of an output variable as `RESULT`.
- Change the name of an input variable to `RESULT`.
- Throughout your code change the variable name from `RESULT` to the input name.

Enhancements to Object-Oriented Programming Capabilities

The Symbolic Math Toolbox product uses some object-oriented programming features to implement symbolic objects. Major enhancements to object-oriented programming capabilities enable easier development and maintenance of large applications and data structures. For a full description of object-oriented features, see the MATLAB Object-Oriented Programming documentation.

Compatibility Considerations

It is no longer possible to add methods to `@sym` by creating a `@sym` directory containing custom methods.

For an empty `x`, `sym(x)` returns a symbolic object of the same size as `x`. In previous releases, `sym(x)` returned a symbolic object of size 0-by-0 for an empty `x`.

`generate::MATLAB` Function Converts MuPAD Expressions to MATLAB Code

The new `generate::MATLAB` command converts MuPAD expressions, equations, and matrices to MATLAB formatted character vectors.

MuPAD `IgnoreAnalyticConstraints` Option Specifies That Core Functions Apply Common Algebraic Assumptions to Simplify Results

The new `IgnoreAnalyticConstraints` option allows the use of a set of simplified mathematical rules when solving equations, simplifying expressions, or integrating. For example, this option applies practical, but not generally correct rules for combining

logarithms: $\ln(a) + \ln(b) = \ln(a \cdot b)$

MuPAD Outputs Contain Abbreviations for Better Readability

The new default format of presenting results enhances readability of long output expressions by using abbreviations.

MuPAD Solver for Ordinary Differential Equations Handles More Equation Types

The solver now can handle more than 200 additional types of second-order ordinary differential equations. The solver demonstrates improved performance.

MuPAD `limit` Function Now Can Compute Limits for Piecewise Functions

The enhanced `limit` function computes limits of piecewise functions including bidirectional and one-sided limits.

New and Improved MuPAD Special Functions

MuPAD includes the following new special functions:

- `laguerreL` represents Laguerre's L function.
- `erfc(x,n)` returns iterated integrals of the complementary error function.
- `meijerG` represents the Meijer G function.

The `hypergeom` special function demonstrates better performance.

New Calling Syntax for Test Report Function `prog::tcov`

The `prog::tcov` function that inspects the data collected during the code execution has the new syntax and set of options.

Compatibility Considerations

The new syntax is not valid in MuPAD versions earlier than 5.2. MuPAD 5.2 does not support the earlier syntax.

New Demos

To see new demos that use MuPAD Notebook Interface, type `mupadDemo` at the MATLAB command line or click MuPAD Notebooks Demo.

R2008b

Version: 5.1

Bug Fixes

R2008a+

Version: 5.0

Bug Fixes

R2007b+

Version: 4.9

New Features

Bug Fixes

Compatibility Considerations

MuPAD Engine Replaces Maple Engine

The default Symbolic Math Toolbox engine is now the MuPAD engine. For more information, see the MuPAD in Symbolic Math Toolbox chapter in the Symbolic Math Toolbox User's Guide.

Compatibility Considerations

The new engine causes many computed results to differ from those returned by previous versions of Symbolic Math Toolbox software.

General Differences

- Many computations return in a permuted order (such as $a + b$ instead of $b + a$).
- Some computations return in a different, mathematically equivalent form (such as $(\cos(x))^2$ instead of $1 - (\sin(x))^2$).
- `diff(dirac(t))` returns `dirac(t,1)` instead of `dirac(1,t)`.
- `sym(x, 'f')` no longer produces character vectors of the form `hex digits*2^n`. Instead the character vectors have the form $(2^{e+N} \cdot 2^{(e-52)})$, where N and e are integers.
- For toolbox calculations, some symbols can only be used as symbolic variables, and not in character vectors: `E`, `I`, `D`, `O`, `beta`, `zeta`, `theta`, `psi`, `gamma`, `Ci`, `Si`, and `Ei`. This is because those symbols represent MuPAD reserved words, and are interpreted as the MuPAD word if you pass them as character vectors. The words `Ci`, `Si`, `Ei` represent special mathematical functions: the cosine integral, sine integral, and exponential integral respectively.
- Error and warning message IDs may have changed.
- Performance of numerical integration is slower than in previous versions.
- Subexpressions, calculated by the `subexpr` function, may be different than in previous versions.
- The `pretty` function no longer uses partial subexpressions (with syntax `%n`).

Calculus

- `Int` no longer evaluates some integrals, including many involving Bessel functions.
- `symsum(sin(k*pi)/k, 0, n)` no longer evaluates to `pi`.

Linear Algebra

- The output of `colspace` may differ from previous versions, but it is mathematically equivalent.
- The `eig` function may return eigenvalues in a different order than previous versions. Expressions returned by `eig` may be larger than in previous versions.
- The `jordan` function may return diagonal subblocks in a different order than previous versions.
- `svd` may return singular values in a different order than previous versions.

Simplification

- The `coeffs` function may return multivariable terms in a different order than in previous versions.
- The `expand` function may return some trig and exponential expressions differently than in previous versions.
- The `simplify` function involving radicals and powers make fewer assumptions on unknown symbols than in previous versions.
- The `subexpr` function may choose a different subexpression to be the common subexpression than in previous versions.
- Subexpressions no longer have partial subexpressions (previous syntax `%n`).
- The `solve` function returns solutions with higher multiplicity only when solving a single polynomial.
- $\operatorname{acot}(-x) = -\operatorname{acot}(x)$ instead of $\pi - \operatorname{acot}(x)$ as in previous versions.
- $\operatorname{acoth}(-x) = -\operatorname{acoth}(x)$ instead of $2*\operatorname{acoth}(0) - \operatorname{acoth}(x)$ as in previous versions.
- The `simple` function has several differences:
 - The 'how' value `combine(trig)` has been replaced with `combine(sincos)`, `combine(sinhcosh)`, and `combine(ln)`.
 - The 'how' values involving `convert` have been replaced with `rewrite`.
 - A new 'how' value of `mlsimplify(100)` indicates the MuPAD function `Simplify(...,Steps=100)` simplified the expression.
 - Simplifications such as $(\sin(x)^2)^{(1/2)}$ to $\sin(x)$ are no longer performed, since the MuPAD language is careful not to make assumptions about the sign of $\sin(x)$.

Conversion

- Arithmetic involving the `vpa` function uses the current number of digits of precision. Variable precision arithmetic may have different rounding behaviors, and answers may differ in trailing digits (trailing zeros are now suppressed).
- The `char` function returns character vectors using MuPAD syntax instead of Maple™ syntax.
- Testing equality does not compare character vectors as in previous versions; the symbolic engine equality test is used.
- Saving and loading symbolic expressions is compatible with previous versions, except when the symbolic contents use syntax or functions that differ between Maple or MuPAD engines. For example, suppose you save the symbolic object `sym('transform::fourier(f,x,w)')`, which has MuPAD syntax. You get a MATLAB error if you try to open the object while using a Maple engine.
- LaTeX output from the `latex` function may look different than before.
- C and Fortran code generated with the `ccode` and `fortran` functions may be different than before. In particular, generated files have intermediate expressions as “optimized” code. For more information, see the Generating C or Fortran Code section of the User's Guide.
- pretty output may look different than before.

Equation Solving

- `solve` returns solutions with higher multiplicity only when solving a single polynomial.
- `solve` may return a different number of solutions than before.
- Some calls to `dsolve` that used to return results involving `lambertw` now return no solution.
- `dsolve` can now use the variable `C`.
- Some `dsolve` results are more complete (more cases are returned).
- Some `dsolve` results are less complete (not all previous answers are found).
- `finverse` may be able to find inverses for different classes of functions than before.
- When `finverse` fails to find an explicit inverse, it produces different output than before.

Transforms

- Fourier and inverse Fourier transforms return the MuPAD form `transform::fourier` when they cannot be evaluated. For example,

```
h = sin(x)/exp(x^2);
FF = fourier(h)
```

```
FF =
transform::fourier(sin(x)/exp(x^2), x, -w)
```

The reason for this behavior is the MuPAD definition of Fourier transform and inverse Fourier transform differ from their Symbolic Math Toolbox counterparts by the sign in the exponent:

	Symbolic Math Toolbox definition	MuPAD definition
Fourier transform	$F(w) = \int_{-\infty}^{\infty} f(x)e^{-iwx} dx$	$F(w) = \int_{-\infty}^{\infty} f(x)e^{iwx} dx$
Inverse Fourier transform	$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(w)e^{iwx} dw$	$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(w)e^{-iwx} dw$

- Several Fourier transforms can no longer be calculated, especially those involving Bessel functions.
- `ztrans` and `iztrans` may return more complicated expressions than before.

Special Mathematical Functions

- The three-parameter Riemann Zeta function is no longer supported.
- `heaviside(0) = 0.5`; in previous versions it was undefined.

maple

- The `maple`, `mhelp`, and `procread` functions error, unless a Maple engine is installed and selected with `symengine`.

New MuPAD Language and Libraries Supplant Extended Symbolic Math Toolbox Software

The functionality of the MuPAD language, together with the included libraries, goes far beyond that of the previous Symbolic Math Toolbox software. However, it is not identical

to that of the previous Extended Symbolic Math Toolbox™ software. The differences between these software packages are beyond the scope of these release notes.

You can access the MuPAD language in several ways:

- To learn the commands, syntax, and functionality of the language, use the MuPAD Help browser, or read the Tutorial.
- Use a MuPAD notebook, which contains an integrated help system for the language syntax.
- Use the new `evalin` function or `feval` function to access the MuPAD language at the MATLAB command line. More detail is available in the Calling Built-In MuPAD Functions from the MATLAB Command Window section of the User's Guide.

New MuPAD Help Viewer (GUI)

The MuPAD help viewer contains complete documentation of the MuPAD language, and of the MuPAD Notebook Interface. For more information, see the Getting Help for MuPAD section of the User's Guide.

New MuPAD Notebook Interface (GUI)

A MuPAD notebook is an interface for performing symbolic math computations with embedded math notation, graphics, animations, and text. It also enables you to share, document, and publish your calculations and graphics. For example, the MuPAD help viewer is essentially a special MuPAD notebook. For more information, see the Calculating in a MuPAD Notebook section of the User's Guide.

New MuPAD Editor and Debugger (GUI)

The MuPAD Editor GUI enables you to write custom symbolic functions and libraries in the MuPAD language. The Debugger enables you to test your code. For more information, consult the MuPAD help viewer.

New Functionality for Communication Between MATLAB Workspace and MuPAD

Function	Use
<code>doc(symengine,...)</code>	Access the MuPAD Help browser.
<code>evalin(symengine,...)</code>	Use MuPAD functionality in the MATLAB workspace.
<code>feval(symengine,...)</code>	Use MuPAD functionality in the MATLAB workspace.
<code>getVar</code>	Copy expressions residing in a MuPAD notebook into the MATLAB workspace.
<code>mupad</code>	Launch a MuPAD notebook .
<code>mupadwelcome</code>	Access MuPAD GUIs .
<code>reset(symengine,...)</code>	Clear the MuPAD engine for the MATLAB workspace.
<code>setVar</code>	Copy expressions residing in the MATLAB workspace into a MuPAD notebook.
<code>symvar</code>	Produce a list of symbolic objects in an expression.

For more information, see the Integration of MuPAD and MATLAB section of the User's Guide.

New `symengine` Command for Choosing a Maple Engine

If you own a compatible version of a Maple software, you can choose to have Symbolic Math Toolbox software use the Maple engine instead of a MuPAD engine. You might want to do this if you have existing Maple programs. Choose the engine by entering `symengine` at the MATLAB command line; this brings up a GUI for making your choice.

New `matlabFunction` Generates MATLAB Functions

The new `matlabFunction` generates MATLAB functions from symbolic expressions. `matlabFunction` writes the generated code to a file or creates a function handle. You can use the generated function handles and files in any MATLAB installation, even those without a Symbolic Math Toolbox license. For more information, see Generating MATLAB Functions in the User's Guide.

R2008a

Version: 3.2.3

Bug Fixes

R2007b

Version: 3.2.2

Bug Fixes

R2007a

Version: 3.2

New Features

Bug Fixes

Maple10 Access Added for Linux 64-bit Processors and Intel Macintosh Platforms

MATLAB now supports Maple Version 10 on 32-bit Windows, 32- and 64-bit Linux® platforms, and the Intel® and PowerPC® Macintosh platforms.

R2006b

Version: 3.1.5

New Features

Bug Fixes

Compatibility Considerations

Change in call to code generation package using the maple function

Calling a function in code generation package using Maple software now requires you to explicitly include the package name. For example,

```
maple('codegen[fortran](x^2-4)');
```

The generated code output using these methods is unaffected by this change.

Compatibility Considerations

In previous versions, functions in the code generation package of Maple software were made automatically available using the Maple `with` command, and did not require the package name. For example

```
maple('fortran(x^2-4)');
```

This sometimes caused a conflict when assigning to Maple variables having the same name as a function in the code generation package.