Release Notes for Symbolic Math Toolbox™

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Contents

R2012b

Version: 5.9
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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`

Compatibility Considerations: Yes

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`

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Old syntax of `taylor`	Errors	New calling syntax	Update all instances of `taylor` function calls using the new syntax.
`char(A,d)`	Errors	`char(A)`	Replace all instances of `char(A,d)` with `char(A)`.
`poly`	Warns	`charpoly`	Replace all instances of `poly` with `charpoly`.
`emlBlock`	Warns	`matlabFunctionBlock`	Replace all instances of `emlBlock` with `matlabFunctionBlock`.
Ability to create links from MuPAD notebooks to MuPAD documentation pages	Not available	Nothing	No replacement
`openmuphlp`	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 (`.mu` 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**

`psi(k0:k1,X)`	Errors	`psi(k,X)`, where `k` is a scalar specifying the `k`th derivative of `psi` at the elements of `X`.	Replace all instances of `psi(k0:k1,X)` with `psi(k,X)`, where `k` is a scalar. To modify your code, loop through the values `k0:k1`. For example:
			
			`for k = k0:k1`
			`  Y(:,k) = psi(k,X);`
			`end`
			
			In a future release, `size(Y)` will be `size(X)`. Modify any code that depends on `size(Y)`.

`sqrt` target of the MuPAD `simplify` function	Errors	MuPAD `radsimp` or `simplifyRadical`	Replace all instances of `simplify` function calls involving the `sqrt` target with radsimp or simplifyRadical. Alternatively, replace these calls with simplify function calls without targets.
`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`	Warns	MuPAD `fourier`	Replace all instances of `transform::fourier` with fourier.
MuPAD `transform::invfourier`	Warns	MuPAD `ifourier`	Replace all instances of `transform::invfourier` with ifourier.

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

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD `transform::invztrans::addpattern`	Warns	MuPAD `iztrans::addpattern`	Replace all instances of `transform::invztrans::addpattern` with iztrans::addpattern.
MuPAD `prog::calledFrom`	Warns	`context(hold(procname))`	Replace all instances of `prog::calledFrom()` with context`(`hold`(procname))`.
MuPAD `prog::calltree`	Warns	`prog::trace`	Use prog::trace instead of `prog::calltree`.
MuPAD `prog::error`	Warns	`getlasterror`	Use getlasterror instead of `prog::error`.
MuPAD `prog::memuse`	Warns	`prog::trace(Mem)`or `bytes()`	Use prog::trace`(Mem)` or bytes`()` instead of `prog::memuse`.
MuPAD `prog::testfunc`	Warns	`print(Unquoted, "...")`	Use print`(Unquoted, "...")` instead of `prog::testfunc`.
MuPAD `prog::testmethod`	Warns	`prog::test(..., Method = myTestMethod)`	Use prog::test`(..., Method = myTestMethod)` instead of `prog::testmethod`.
MuPAD `prog::testnum`	Warns	Nothing	No replacement
Dynamic modules for MuPAD, including the `module`, `external`, and `Pref::unloadableModules` functions and all functions of the `module` library	Warns	Nothing	No replacement

R2012a

Version: 5.8
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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

`psi(k0:k1,X)`	Warns	`psi(k,X)` where `k` is a scalar specifying the `k`th derivative of `psi` at the elements of `X`.	Replace all instances of `psi(k0:k1,X)` with `psi(k,X)`, where `k` is a scalar. To modify your code, loop through the values `k0:k1`. For example:
			
			`for k = k0:k1`
			`  Y(:,k) = psi(k,X);`
			`end`
			
			In the future, `size(Y)` will be `size(X)`. Modify any code that depends on `size(Y)`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`sqrt` target of the MuPAD `simplify` function	Warns	MuPAD `radsimp` or `simplifyRadical`	Replace all instances of `simplify` function calls involving the `sqrt` target with `radsimp` or `simplifyRadical`. Alternatively, replace these calls with `simplify` function calls without targets.
`cos`, `sin`, `exp`, and `ln` targets of the MuPAD `simplify` function	Warns	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 `frame` function	Errors	Nothing	No replacement

R2011b

Version: 5.7
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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

R2011a

Version: 5.6
New Features: Yes
Bug Fixes: Yes

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`, `symdiffdemo`, or `symintdemo` in the MATLAB Command Window.

Functionality Being Removed or Changed

Compatibility Considerations: Yes

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
MuPAD `matchlib::analyze`	Errors	MuPAD `prog::exprtree`	To visualize expressions, use `prog::exprtree`.
MuPAD `prog::testcall`	Errors	Nothing	No replacement
MuPAD `prog::testerrors`	Errors	Nothing	No replacement
Old syntax of MuPAD `prog::getOptions`	Errors	The new syntax	Update all instances of `prog::getOptions` calls using the new syntax.
Old syntax of MuPAD `prog::trace`	Errors	The new syntax	Update all instances of `prog::trace` calls using the new syntax.

R2010b

Version: 5.5
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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 polynomials 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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`diff` and `int` methods for inputs of the `char` type	Warns	`sym`	Use the `sym` method instead.
MuPAD `matchlib::analyze`	Warns	MuPAD `prog::exprtree`	To visualize expressions, use `prog::exprtree`.
MuPAD `prog::testcall`	Warns	None	No replacement
MuPAD `prog::testerrors`	Warns	None	No replacement
The following options in MuPAD `prog::trace`: • `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: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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 new`All` 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

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

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

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

Compatibility Considerations: Yes

`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

Compatibility Considerations: Yes

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*2^(e - 52))` or `-(2^e + N*2^(e - 52))` , with integers for `N` and `e`, and `N    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

Compatibility Considerations: Yes

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

R2009a

Version: 5.2
New Features: Yes
Bug Fixes: Yes

dsolve Accepts the New Option IgnoreAnalyticConstraints

Compatibility Considerations: Yes

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

Compatibility Considerations: Yes

`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

Compatibility Considerations: Yes

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 strings.

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

Compatibility Considerations: Yes

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
New Features: No
Bug Fixes: Yes

R2008a+

Version: 5.0
New Features: No
Bug Fixes: Yes

R2007b+

Version: 4.9
New Features: Yes
Bug Fixes: Yes

MuPAD Engine Replaces Maple Engine

Compatibility Considerations: Yes

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 strings of the form `hex digits*2^n`. Instead the strings have the form `(2^e+N*2^(e-52))`, where `N` and `e` are integers.
- For toolbox calculations, some symbols can only be used as symbolic variables, and not in strings: `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 strings. 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.
- `acot(-x) = -acot(x)` instead of `pi - acot(x)` as in previous versions.
- `acoth(-x) = -acoth(x)` instead of `2*acoth(0) - 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 strings using MuPAD syntax instead of Maple™ syntax.
- Testing equality does not compare strings 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
`doc(symengine,...)`	Access the MuPAD Help browser.
`evalin(symengine,...)`	Use MuPAD functionality in the MATLAB workspace.
`feval(symengine,...)`	Use MuPAD functionality in the MATLAB workspace.
`getVar`	Copy expressions residing in a MuPAD notebook into the MATLAB workspace.
`mupad`	Launch a MuPAD notebook .
`mupadwelcome`	Access MuPAD GUIs .
`reset(symengine,...)`	Clear the MuPAD engine for the MATLAB workspace.
`setVar`	Copy expressions residing in the MATLAB workspace into a MuPAD notebook.
`symvar`	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
New Features: No
Bug Fixes: Yes

R2007b

Version: 3.2.2
New Features: No
Bug Fixes: Yes

R2007a

Version: 3.2
New Features: Yes
Bug Fixes: Yes

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: Yes
Bug Fixes: Yes

Change in call to code generation package using the maple function

Compatibility Considerations: Yes

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.