Symbolic Math Toolbox™ Release Notes

MATLAB®

MathWorks®

How to Contact MathWorks

Latest news:	www.mathworks.com
Sales and services:	www.mathworks.com/sales_and_services
User community:	www.mathworks.com/matlabcentral
Technical support:	www.mathworks.com/support/contact_us
Phone:	508-647-7000

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

Symbolic Math Toolbox™ Release Notes

Contents

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2009b

R2009a

R2008b

R2008a+

R2007b+

R2008a

R2007b

R2007a

R2006b

R2022b

Symbolic Matrix Variables: double, vpa, and sym functions accept symbolic matrix variables as input arguments

The `double`, `vpa`, and `sym` functions now accept symbolic matrix variables of type `symmatrix` as input arguments.

Symbolic Equations or Conditions: subs, lhs, and rhs functions accept symbolic equations or conditions involving symbolic matrix variables and functions

The `subs`, `lhs`, and `rhs` functions now accept symbolic equations or conditions involving symbolic matrix variables and matrix functions as input arguments. The input data type can be `symmatrix` or `symfunmatrix`.

Physical Units: New physical constants added to symbolic units

`symunit` includes these new physical constants:

- `R_K` – von Klitzing constant
- `alpha` – fine-structure constant
- `g_e` – electron g-factor
- `g_p` – proton g-factor
- `gamma_e` – electron gyromagnetic ratio
- `gamma_p` – proton gyromagnetic ratio

For more information, see “Units and Unit Systems List”.

Live Editor Display: Display symbolic variables with signs as subscripts and superscripts

You can now display −, +, ±, and # signs as subscripts and superscripts of symbolic variables. In MATLAB® Live Editor, create symbolic variables with those signs as suffixes. For examples, see “Add Subscripts, Superscripts, and Accents to Symbolic Variables in the Live Editor”.

Functionality being removed or changed

separateUnits returns simplified expression after separating units
Behavior change

Starting in R2022b, `separateUnits` returns a simplified expression after separating units. For example, separate the units from an expression `expr`, and return the remaining expression as `Data` and the units as `Units`.

```
u = symunit;
syms t tau;
t_s = t*u.s;
t_0 = 0.01*u.s;
tau_s = tau*u.s;
a = 2*u.V;
```

```
b = 6*u.V;
expr = a + b*exp(-(t_s-t_0)/tau_s)
[Data,Units] = separateUnits(expr)

expr =
2*[V] + 6*exp((((1/100)*[s] + (-t)*[s])/tau)*(1/[s]))*[V]

Data =
6*exp(-(100*t - 1)/(100*tau)) + 2

Units =
[V]
```

In previous releases, `separateUnits` returns an unsimplified expression: `Data = 6*exp((((1/100)*[s] - t*[s])/tau)*(1/[s])) + 2`.

Return the names of all symbolic objects in the MATLAB workspace
Behavior change

Starting in R2022b, you can list the names of all symbolic objects in the MATLAB workspace using `syms` and return them as a cell array using `S = syms`.

The returned names of all symbolic objects now include all symbolic matrix variables and matrix functions of type `symmatrix` and `symfunmatrix` in the MATLAB workspace. In previous releases, `syms` only returns the names of symbolic scalar variables, functions, and arrays of type `sym` and `symfun`.

convertMuPADNotebook will be removed
Still runs

The `convertMuPADNotebook` function will be removed in a future release. Convert your MuPAD® notebooks to MATLAB live scripts now, and use the MATLAB Live Editor instead.

R2022a

Symbolic Matrix Functions: Perform parameter-dependent linear algebra calculations in compact matrix notation

You can use a new symbolic matrix function data type to represent parameter-dependent functions that operate on scalars, vectors, and matrices in compact matrix notation. Symbolic matrix functions offer a concise typeset display and show mathematical formulas with more clarity. Using them, you can take vector- and matrix-based functions from textbooks, enter them in Symbolic Math Toolbox, and perform linear algebra calculations.

The `syms` and `symfunmatrix` functions create symbolic matrix functions. To convert a symbolic matrix function of type `symfunmatrix` to a symbolic function of type `symfun`, use `symfunmatrix2symfun`.

To show all the functions in Symbolic Math Toolbox that accept symbolic matrix functions as input, use the command `methods symfunmatrix`.

norm Function: Compute the Frobenius norm of a symbolic array

The `norm` function now accepts a symbolic multidimensional array as its input argument. Use the syntax `norm(X,"fro")` to return the Frobenius norm of a symbolic array `X`.

R2021b

Next Step Suggestions: Use suggestion menus for symbolic workflows in Live Editor

When you run code that generates symbolic output, Live Editor provides a context menu with next step suggestions that are specific to the output. To open the suggestion menu, you can right-click on the symbolic output, or you can hover over the symbolic output and click the three-dot icon ⋮. For examples, see Next Step Suggestions for Symbolic Workflows in Live Editor.

Compact Display: Command Window displays symbolic values in table, cell, and structure arrays explicitly

The Command Window now displays symbolic values in table, cell, and structure arrays explicitly if they fit into the available width of the Command Window. If the scalar symbolic value is too long, then the display is truncated with an ellipsis at the end.

For example, find the exact and numeric solutions of the following equations and display the results in a table in the Command Window.

```
syms x
eq1 = x^2-5*x+6;
eq2 = 4*x^2-1;
ExactSols = [solve(eq1)'; solve(eq2)'];
NumericSols = [vpasolve(eq1)'; vpasolve(eq2)'];
T = table(Equations,ExactSols,NumericSols)
```

```
T =

  2×3 table

       Equations            ExactSols      NumericSols
    ________________    ___________    ___________

    x^2 - 5*x + 6       2       3      2.0     3.0
    3*x^2 - 7*x - 13    -1/2    1/2    -0.5    0.5
```

Display the subexpressions of the following symbolic expression by using the `children` function.

```
syms x y
subexpr = children(x^2 + x*y + y^2)
```

```
subexpr =

  1×3 cell array

    {[x*y]}    {[x^2]}    {[y^2]}
```

Display the solution of an equation with parameters and conditions of the solution by using the `solve` function and specifying `'ReturnConditions'` as `true`.

```
syms x
S = solve(sin(x) == 0,'ReturnConditions',true)
```

```
S =

  struct with fields:
```

```
         x: pi*k
parameters: k
conditions: in(k, 'integer')
```

Symbolic Matrix Variables: gradient, inv, latex, and subs functions accept symbolic matrix variables as input arguments

The `gradient`, `inv`, `latex`, and `subs` functions now accept symbolic matrix variables of `symmatrix` data type as input arguments.

svd Function: Return singular values as a column vector or diagonal matrix

The `svd` function now accepts the `outputForm` input argument. You can specify `outputForm` as `'vector'` or `'matrix'` to return singular values as a column vector or diagonal matrix.

Functionality being removed or changed

svd and eig return exact solutions in terms of the root function
Behavior change

Starting in R2021b, if `sigma = svd(A)` cannot find the exact singular values in terms of symbolic numbers, then it returns the exact singular values in terms of the `root` function instead. In previous releases, `sigma = svd(A)` returns the singular values as floating-point numbers. To numerically approximate the exact singular values, use `vpa`. For more information, see Compatibility Considerations.

Starting in R2021b, if `eig(A)` cannot find the exact eigenvalues in terms of symbolic numbers, then it returns the exact eigenvalues in terms of the `root` function instead. In previous releases, `eig(A)` returns the eigenvalues as floating-point numbers. To numerically approximate the exact eigenvalues, use `vpa`. For more information, see Compatibility Considerations.

vpasolve(eqns,vars) returns all solutions when vars is declared as a symbolic matrix
Behavior change

Starting in R2021b, `vpasolve(eqns,vars)` returns all solutions when `vars` is declared as a symbolic matrix. In previous releases, if `vars` is a symbolic matrix, then `vpasolve(eqns,vars)` treats `vars` as a column vector and it only returns the solutions for the first column of `vars`. For more information, see Compatibility Considerations.

syms clears assumptions on symbolic variables when creating symbolic functions
Behavior change

Starting in R2021b, when creating symbolic functions with symbolic variables as input arguments, `syms` clears all previously set assumptions on the symbolic variables. For example:

```
syms x y z real
syms t real positive
assumptions
```

```
ans =

[in(x, 'real'), in(y, 'real'), in(z, 'real'), 0 < t]
```

```
syms f(x,y,z,t)
assumptions
```

```
ans =

Empty sym: 1-by-0
```

To set assumptions on the input arguments of symbolic functions, create the symbolic functions first, and then set the assumptions on the symbolic variables (or recreate the symbolic variables with set assumptions). For example:

```
syms g(x,y,z,t)
assume(x,'real')
assume(t,'positive')
assumptions
```

```
ans =

[in(x, 'real'), 0 < t]
```

```
syms y z real
assumptions
```

```
ans =

[in(x, 'real'), in(y, 'real'), in(z, 'real'), 0 < t]
```

R2021a

Symbolic Matrix Variables: Perform linear algebra calculations in compact matrix notation

Represent matrices and vectors in compact matrix notation with a new symbolic matrix variable data type. Symbolic matrix variables offer a concise typeset display and show mathematical formulas with more clarity. Using them, you can take matrix- and vector-based expressions from textbooks, enter them in Symbolic Math Toolbox, and perform linear algebra calculations.

The `syms` and `symmatrix` functions create symbolic matrix variables. To convert a symbolic matrix variable to an array of symbolic scalar variables, use `symmatrix2sym`. For an example, see Create Symbolic Matrix Variables.

Partial Differential Equations (PDEs): Convert symbolic PDEs into the computational form required by Partial Differential Equation Toolbox

Partial Differential Equation Toolbox™ solves systems of equations of the form

$$\mathbf{m}\frac{\partial^2 \mathbf{u}}{\partial t^2} + \mathbf{d}\frac{\partial \mathbf{u}}{\partial t} - \nabla \cdot (\mathbf{c} \otimes \nabla \mathbf{u}) + \mathbf{a}\mathbf{u} = \mathbf{f}$$

The `pdeCoefficients` and `pdeCoefficientsToDouble` functions convert symbolic PDEs into this form and extract the coefficients into a structure that can be used by `specifyCoefficients` (Partial Differential Equation Toolbox). For an example, see Solve Partial Differential Equation of Nonlinear Heat Transfer.

Maximum and Minimum Elements: Find maximum and minimum elements of an array of symbolic expressions

The `max` and `min` functions can now operate on symbolic expressions involving symbolic variables, where they threw an error in previous releases.

The `max` function returns the maximum elements, and the `min` function returns the minimum elements of an array of symbolic expressions.

C Code Generation: Generate C code faster with MATLAB Coder from MATLAB functions converted using matlabFunction

To generate optimized C or C++ code faster from a large symbolic expression, use the MATLAB Coder™ app. This way, the generated code is better integrated into the MATLAB ecosystem.

First, convert the symbolic expression to a deployable MATLAB function using `matlabFunction`. Then, generate C or C++ code from the MATLAB function using the MATLAB Coder app. For an example, see Generate C Code from Symbolic Expressions Using the MATLAB Coder App.

R2020b

vpasum Function: Evaluate numerical summation using variable precision

`vpasum` returns the numerical summation of a symbolic series using variable precision.

log2 Function: Evaluate mantissas and exponents

The symbolic `log2` function can now return two output arguments for the mantissas and exponents of symbolic expressions. For example, `[F,E] = log2(X)` returns arrays `F` and `E` of mantissas and exponents, respectively, such that `X = F.*(2.^E)`.

cumsum and cumprod Functions: Ignore NaNs using 'omitnan'

`cumsum` and `cumprod` can exclude `NaN`s when evaluating the cumulative sum and cumulative product of an array that contains `NaN`s. You can use the option `'omitnan'` to exclude `NaN`s and `'includenan'` to include `NaN`s.

symtrue and symfalse Functions: Create symbolic matrices and arrays of symbolic logical constants

`symtrue` and `symfalse` can now create matrices and multidimensional arrays that contain symbolic logical constants `symtrue` and `symfalse`.

Function Input Arguments: Support for multidimensional arrays of symbolic expressions

The following functions now accept multidimensional arrays of symbolic expressions as their input arguments: `sum`, `cumsum`, `prod`, `cumprod`, `sort`, `min`, `max`, `mean`, `std`, and `var`.

Functionality being removed or changed

children now returns a nonnested or nested cell array
Behavior change

In versions before R2020b, the syntax `subexpr = children(expr)` returns a vector `subexpr` that contains the child subexpressions of the scalar symbolic expression `expr`. The syntax `subexpr = children(A)` returns a nonnested cell array `subexpr` that contains the child subexpressions of the symbolic array `A`.

Starting in R2020b, the syntax `subexpr = children(expr)` returns `subexpr` as a cell array instead of a vector, and the syntax `subexpr = children(A)` returns `subexpr` as a nested cell array instead of a nonnested cell array. You can use `subexpr = children(expr,ind)` to index into the returned cell arrays of subexpressions. You can also unnest and access the elements of a cell array by indexing into the cell array using curly braces. To convert `subexpr` from a nonnested cell array to a vector, you can use the command `[subexpr{:}]`

mod no longer finds the modulus for each coefficient of a symbolic polynomial
Behavior change

Starting in R2020b, `mod` no longer finds the modulus for each coefficient of a symbolic polynomial. Instead, `mod(a,b)` returns an unevaluated symbolic expression if `a` is a symbolic polynomial and `b` is a real number. To find the modulus for each coefficient of the polynomial `a`, use `[c,t] = coeffs(a); sum(mod(c,b).*t)`. You can now create periodic symbolic functions by defining the periodicity using `mod`. For example, see Create Periodic Sawtooth Waves.

R2020a

Live Editor Tasks: Interactively solve equations, simplify symbolic expressions, and generate MATLAB code in a live script

Use Live Editor with Symbolic Math Toolbox for two new tasks:

- **Solve Symbolic Equation** — Find solutions of symbolic equations.
- **Simplify Symbolic Expression** — Simplify symbolic expressions.

The tasks also automatically generate code that becomes part of your live script.

To use tasks in the Live Editor, on the **Live Editor** tab, in the **Task** menu, select a task. Alternatively, in a code block in a live script, begin typing the task name and select the task from the suggested command completions. For more information about Live Editor tasks generally, see Add Interactive Tasks to a Live Script (MATLAB).

Number Theory: Evaluate the Euler phi function, Jacobi symbol, and find rational fraction approximations and primitive roots

- `eulerPhi` evaluates the Euler totient function.
- `jacobiSymbol` evaluates the Jacobi symbol.
- `rat` returns the rational fraction approximation of a symbolic input.
- `isPrimitiveRoot` checks whether an integer is a primitive root modulo another integer.

Differential Equations: Return solutions of differential equations in implicit form or truncated series expansion

With `dsolve`:

- Use the `'Implicit'` name-value pair to return implicit solutions of differential equations.
- Use the `'ExpansionPoint'` and `'Order'` name-value pairs to specify the expansion point and truncation order of Puiseux series solutions of differential equations.

Physical Units: New physical constants and application of 2019 SI redefinition to symbolic units

- New physical constants added to `symunit`:
  - `G_0` – conductance
  - `e_0` – vacuum electric permittivity or electric constant
  - `F_c` – Faraday constant
  - `mu_0` – vacuum magnetic permeability
  - `sigma` – Stefan-Boltzmann constant
  - `phi_0` – magnetic flux quantum
  - `R_c` – molar gas constant
  - `m_p` – proton mass
  - `R_inf` – Rydberg constant

For more information, see Units and Unit Systems List.

- Symbolic units have been updated using the 2019 SI redefinition.

Symbolic Constants: Use symbolic logical constants for true and false conditions

- `symtrue` returns the symbolic logical constant for the true condition.
- `symfalse` returns the symbolic logical constant for the false condition.

Functionality being removed or changed

MuPAD notebook has been removed

Errors

MuPAD notebook has been removed. Use MATLAB Live Editor instead. To convert a MuPAD notebook file to a MATLAB live script file, see `convertMuPADNotebook`.

If you cannot find the Symbolic Math Toolbox equivalent for MuPAD functionality, contact MathWorks Technical Support. To access MuPAD documentation in previous releases, see Archived MathWorks Documentation.

sym('pi') now creates a symbolic variable named pi

Behavior change

`sym('pi')` now creates a symbolic variable named `pi`. To create a symbolic number that represents the mathematical constant `pi`, use `sym(pi)`.

mod will no longer find the modulus for each coefficient of a symbolic polynomial

Behavior change in future release

In a future release, `mod` will no longer find the modulus for each coefficient of a symbolic polynomial. Instead, `mod(a,b)` will return an unevaluated symbolic expression if `a` is a symbolic polynomial and `b` is a real number. To find the modulus for each coefficient of the polynomial `a`, use `[c,t] = coeffs(a); sum(mod(c,b).*t)`.

R2019b

Calculus: Apply integration by parts and integration by substitution to integrals

- `integrateByParts` applies integration by parts to integrals.
- `changeIntegrationVariable` applies integration by substitution to integrals.

Integral: Return unevaluated integral

`int( ___ ,'Hold',true)` returns integrals without evaluating them. Use `release` to return the evaluated integrals by ignoring the `'Hold'` option in the `int` function.

Display Formula: Display symbolic formula from string

`displayFormula` displays a symbolic formula from a string in the Live Editor.

Live Editor: Copy and paste symbolic outputs

You can now copy symbolic outputs and paste them as MATLAB code or typeset equations in the Live Editor.

Number Conversion: Convert a decimal number to binary or hexadecimal notation

- `dec2bin` converts a decimal number to a character vector representing a binary number.
- `dec2hex` converts a decimal number to a character vector representing a hexadecimal number.

You can now convert a symbolic number larger than `flintmax` to binary or hexadecimal notation. For example, `dec2hex(sym(2^60))` returns `'1000000000000000'`.

Functionality being removed or changed

mfun and mfunlist have been removed

Errors

`mfun` and `mfunlist` have been removed. Use the appropriate special function instead. For a list of available special functions, see Mathematical Functions.

dsolve and odeToVectorField will no longer support character vector or string inputs

Warns

`dsolve` and `odeToVectorField` will no longer support character vector and string inputs in a future release. Use `sym` objects instead to define and solve differential equations.

R2019a

Animations: Create animations from symbolic expressions

- `fanimator` creates a stop-motion animation object.
- `playAnimation` plays animation objects in a MATLAB figure window.
- `rewindAnimation` rewinds previously played animation objects.
- `writeAnimation` saves animation objects to a GIF or AVI file.
- `animationToFrame` writes animation objects to a structure array of frames.

Symbolic Types: Categorize symbolic objects by type

- `symType` determines the type of a symbolic object.
- `symFunType` determines the functional type of a symbolic object.
- `isSymType` checks whether a symbolic object is of a specific type.
- `hasSymType` checks whether a symbolic object contains subobjects of a specific type.
- `findSymType` finds symbolic subobjects of a specific type.
- `mapSymType` applies a function to symbolic subobjects of a specific type.

Display Output: Change the display of symbolic output

The `sympref` function now accepts the following preferences:

- `'FloatingPointOutput'` displays symbolic numbers in short fixed-decimal format.
- `'PolynomialDisplayStyle'` displays polynomials in ascending or descending order.
- `'MatrixWithSquareBrackets'` displays symbolic matrices with square brackets in Live Scripts.

Scientific Display: Display symbolic variables with accents, subscripts, and superscripts in standard mathematical notation

MATLAB Live Editor now displays symbolic variables, such as $\dot{x}_1$, in standard mathematical notation. Add Subscripts, Superscripts, and Accents to Symbolic Variables by appending suffixes.

Integral Transforms: Evaluate the Hilbert transform and its inverse analytically

- `htrans` returns the Hilbert transform for symbolic expressions.
- `ihtrans` returns the inverse Hilbert transform for symbolic expressions.

hurwitzZeta Function: Evaluate the Hurwitz zeta function analytically

`hurwitzZeta` returns the Hurwitz zeta function for numeric and symbolic inputs.

R2018b

unitConvert Function: Convert physical values between units or unit systems

`unitConvert` converts between units of measurement.

besselh Function: Evaluate the Hankel function analytically

`besselh` returns the Hankel function for symbolic expressions.

nthroot Function: Calculate the nth root of symbolic expressions

`nthroot(x,n)` returns the `n`th root of the symbolic expression `x`. The `nthroot` function is similar to `power`, but returns the root with the complex phase closest to the phase of the expression `x`.

sinc Function: Work with the sinc function analytically

`sinc` returns the sinc function for symbolic expressions.

mathml Function: Generate MathML markup from symbolic expressions

`mathml` generates MathML markup from a symbolic expression.

Variable Assumptions: syms clears assumptions

The `syms` function now clears all assumptions from its variables by default.

Compatibility Considerations

In previous releases, `syms` retained assumptions on cleared variables. Because `syms` now clears assumptions, to retain assumptions on cleared variables, recreate the variables using `sym`. For example:

```
syms x real
assume(x <= 5);
clear x
x = sym('x');
assumptions(x)
```

```
ans =

x <= 5
```

Functionality being removed or changed

symengine, feval, evalin, and read are not recommended for symbolic calculations

Still runs

Symbolic Math Toolbox includes operations and functions for symbolic math expressions that parallel MATLAB functionality for numeric values. Unlike MuPAD functionality, Symbolic Math Toolbox functions enable you to work in familiar interfaces, such as the MATLAB Command Window or Live Editor, which offer a smooth workflow and are optimized for usability.

Therefore, instead of passing `symengine` and MuPAD expressions to `feval`, `evalin`, or `read`, use the equivalent Symbolic Math Toolbox functionality to work with symbolic math expressions. For a list of available functions, see Symbolic Math Toolbox functions list.

To convert a MuPAD notebook file to a MATLAB live script file, see `convertMuPADNotebook`.

If you cannot find the Symbolic Math Toolbox equivalent for MuPAD functionality, contact MathWorks Technical Support.

Although the use of `symengine`, `feval`, `evalin`, and `read` is not recommended for symbolic calculations, there are no plans to remove these functions at this time.

reset is not recommended
Still runs

To update your code, replace any instance of `reset(symengine)` with `clear all`. The `clear all` call closes the MuPAD engine associated with the MATLAB workspace, resets all associated assumptions, and removes all variables, including symbolic objects, from the MATLAB workspace.

Although the use of `reset` is not recommended, there are no plans to remove it at this time.

syms will no longer support clear option
Warns

The syntaxes `syms x clear` and the equivalent `syms('x','clear')` now warn that they will be removed in a future release.

In previous releases, both syntaxes cleared all assumptions applied to `x`. To update your code, call `syms` and specify the variables whose assumptions you want to clear. For example, `syms x` clears all assumptions applied to `x`.

R2018a

Polynomial Operations: Calculate the degree, resultant, and reduction of polynomials

- polynomialDegree returns the degree of a polynomial.
- polynomialReduce reduces or divides polynomials and equations to reduce their order.
- resultant calculates the resultant of two polynomials.

Groebner Basis: Calculate the Groebner basis and eliminate variables from equations

- gbasis calculates the Groebner basis of polynomials.
- eliminate eliminates variables from polynomial equations.

Number Theory: Calculate perfect powers, modular powers, and prime numbers

- nthprime returns the nth prime number.
- powermod returns the modular power a^b mod m.
- factorIntegerPower returns the perfect power factorization.
- nextprime and prevprime now accept double input.
- gcd also returns the linear combination of the input that equals the GCD.

Physical Units: Convert between more unit systems, use more physical dimensions, and display mixed units

- Added unit systems EMU, ESU, and GU. See Units and Unit Systems List.
- unitInfo returns information on the physical dimensions of compound units.
- mixedUnits splits the input quantity into a combination of units.

MATLAB Live Scripts: Convert MuPAD notebooks, which will be removed in a future release, to MATLAB live scripts by using convertMuPADNotebook

Convert MuPAD notebooks to MATLAB live scripts by using convertMuPADNotebook. See Convert MuPAD Notebooks to MATLAB Live Scripts. MuPAD notebooks will be removed in a future release.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD notebooks	Warns	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.
`eval` for symbolic inputs	Still runs	Use `subs` to replace symbolic variables by their values. Use `double` to convert symbolic objects to double.	
Character vectors as inputs to `dsolve`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `dsolve('Dy = y')` with `syms y(t); dsolve(diff(y,t) == y)`.	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.
Character vectors as inputs to `odeToVectorField`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `odeToVectorField('D2y = x')` with `syms y(x); odeToVectorField(diff(y,x,2) == x)`.	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.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations

Character vector inputs to `sym` and `vpa` are restricted to numbers and valid variable names.	Errors	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:
		• `syms x; x + 1`
		instead of
		`sym('x + 1')`
		• `exp(sym(pi))`
		instead of
		`sym('exp(pi)')`
		• `syms f(var1,...varN)`
		instead of
		`f(var1,...varN) = sym('f(var1,...varN)')`
		• `vpa((1 + sqrt(sym(5)))/2)`
		instead of
		`vpa('(1 + sqrt(5))/2')`
		For programmatic workflows, use `str2sym`.
			Replace all instances of character vectors that are not valid variable names and do not define a number. Instead, first create symbolic variables, and then use operations on them. For programmatic workflows, use `str2sym`.

`ezplot`	Still runs	`fplot`, `fimplicit`	Replace `ezplot` with `fplot` for 2-D line plots and with `fimplicit` for 2-D implicit plots.
`ezplot3`	Still runs	`fplot3`	Replace `ezplot3` with `fplot3`.
`ezsurf` and `ezsurfc`	Still runs	`fsurf`	Replace `ezsurf` with `fsurf`. Replace `ezsurfc` with `fsurf(...,'ShowContours','on')`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`ezcontour` and `ezcontourf`	Still runs	`fcontour`	Replace `ezcontour` with `fcontour`. Replace `ezcontourf` with `fcontour(...,'Fill','on')`.
`ezmesh` and `ezmeshc`	Still runs	`fmesh`	Replace `ezmesh` with `fmesh`. Replace `ezmeshc` with `fmesh(...,'ShowContours', 'on')`.
Character vectors as inputs to `solve`	Errors	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)`.	Replace all instances of character vector input. 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.
`mfunlist`	Warns	See Mathematical Functions.	See Mathematical Functions for the list of available special functions.

R2017b

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
`eval` for symbolic inputs	Still runs	Use `subs` to replace symbolic variables by their values. Use `double` to convert symbolic objects to double.	
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 `dsolve`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `dsolve('Dy = y')` with `syms y(t); dsolve(diff(y,t) == y)`.	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.
Character vectors as inputs to `odeToVectorField`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `odeToVectorField('D2y = x')` with `syms y(x); odeToVectorField(diff(y,x,2) == x)`.	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.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations

Character vector inputs to `sym` and `vpa` are restricted to numbers and valid variable names.	Warns	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:
		• `syms x; x + 1`
		instead of
		`sym('x + 1')`
		• `exp(sym(pi))`
		instead of
		`sym('exp(pi)')`
		• `syms f(var1,...varN)`
		instead of
		`f(var1,...varN) = sym('f(var1,...varN)')`
		• `vpa((1 + sqrt(sym(5)))/2)`
		instead of
		`vpa('(1 + sqrt(5))/2')`
			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.

`ezplot`	Still runs	`fplot`, `fimplicit`	Replace `ezplot` by `fplot` for 2-D line plots and by `fimplicit` for 2-D implicit plots.
`ezplot3`	Still runs	`fplot3`	Replace `ezplot3` by `fplot3`.
`ezsurf` and `ezsurfc`	Still runs	`fsurf`	Replace `ezsurf` by `fsurf`. Replace `ezsurfc` by `fsurf(...,'ShowContours','on')`.
`ezcontour` and `ezcontourf`	Still runs	`fcontour`	Replace `ezcontour` with `fcontour`. Replace `ezcontourf` by `fcontour(...,'Fill','on')`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`ezmesh` and `ezmeshc`	Still runs	`fmesh`	Replace `ezmesh` by `fmesh`. Replace `ezmeshc` 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.
`mfunlist`	Warns	See Mathematical Functions.	See Mathematical Functions for the list of available special functions.

R2017a

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 `dsolve`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `dsolve('Dy = y')` with `syms y(t); dsolve(diff(y,t) == y)`.	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.
Character vectors as inputs to `odeToVectorField`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `odeToVectorField('D2y = x')` with `syms y(x); odeToVectorField(diff(y,x,2) == x)`.	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.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations

Character vector inputs to `sym` and `vpa` are restricted to numbers and valid variable names.	Warns	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:
		• `syms x; x + 1`
		instead of
		`sym('x + 1')`
		• `exp(sym(pi))`
		instead of
		`sym('exp(pi)')`
		• `syms f(var1,...varN)`
		instead of
		`f(var1,...varN) = sym('f(var1,...varN)')`
		• `vpa((1 + sqrt(sym(5)))/2)`
		instead of
		`vpa('(1 + sqrt(5))/2')`
			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.

`ezplot`	Still runs	`fplot`, `fimplicit`	Replace `ezplot` by `fplot` for 2-D line plots and by `fimplicit` for 2-D implicit plots.
`ezplot3`	Still runs	`fplot3`	Replace `ezplot3` by `fplot3`.
`ezsurf` and `ezsurfc`	Still runs	`fsurf`	Replace `ezsurf` by `fsurf`. Replace `ezsurfc` by `fsurf(...,'ShowContours','on')`.
`ezcontour` and `ezcontourf`	Still runs	`fcontour`	Replace `ezcontour` with `fcontour`. Replace `ezcontourf` by `fcontour(...,'Fill','on')`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`ezmesh` and `ezmeshc`	Still runs	`fmesh`	Replace `ezmesh` by `fmesh`. Replace `ezmeshc` 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.
`mfunlist`	Warns	See Mathematical Functions.	See Mathematical Functions for the list of available special functions.

R2016b

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 `dsolve`	Still runs	When specifying equations and variables, use symbolic equations and variables instead of character vectors. For example, replace `dsolve('Dy = y')` with `syms y(t); dsolve(diff(y,t) == y)`.	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.
Character vector inputs to `sym` and `vpa` are restricted to numbers and valid variable names.	Warns	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use: • `syms x; x + 1` instead of `sym('x + 1')` • `exp(sym(pi))` instead of `sym('exp(pi)')` • `syms f(var1,...varN)` instead of `f(var1,...varN) = sym('f(var1,...varN)')` • `vpa((1 + sqrt(sym(5)))/2)` instead of `vpa('(1 + sqrt(5))/2')`	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
`ezplot`	Still runs	`fplot`, `fimplicit`	Replace `ezplot` by `fplot` for 2-D line plots and by `fimplicit` for 2-D implicit plots.
`ezplot3`	Still runs	`fplot3`	Replace `ezplot3` by `fplot3`.
`ezsurf` and `ezsurfc`	Still runs	`fsurf`	Replace `ezsurf` by `fsurf`. Replace `ezsurfc` by `fsurf(...,'ShowContours', 'on')`.
`ezcontour` and `ezcontourf`	Still runs	`fcontour`	Replace `ezcontour` with `fcontour`. Replace `ezcontourf` by `fcontour(...,'Fill','on')`.
`ezmesh` and `ezmeshc`	Still runs	`fmesh`	Replace `ezmesh` by `fmesh`. Replace `ezmeshc` 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`.
`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.
`mfunlist`	Warns	See Mathematical Functions.	See Mathematical Functions for the list of available special functions.

R2016a

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 `sym` and `vpa` are restricted to numbers and valid variable names.	Warns	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:
		• `syms x; x + 1`
		instead of
		`sym('x + 1')`
		• `exp(sym(pi))`
		instead of
		`sym('exp(pi)')`
		• `syms f(var1,...varN)`
		instead of
		`f(var1,...varN) = sym('f(var1,...varN)')`
		• `vpa((1 + sqrt(sym(5)))/2)`
		instead of
		`vpa('(1 + sqrt(5))/2')`
			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.

Comparing symbolic and non-symbolic objects by using `isequal`	Returns `0` instead of `1`.	Wrap the non-symbolic object with `sym`. For example, instead of `a=1;isequal(a,1)`, use `isequal(a,sym(1))`.	Symbolic objects are not considered equal to non-symbolic objects.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
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 of character vector inputs, 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`.
`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 Special Functions for the list of available special functions.
`mfunlist`	Warns	See Special Functions.	See Special Functions for the list of available special functions.
`sym(A,set)` and `sym(A,'clear')` where `A` is a symbolic object in the workspace.	Errors	`assume(A,set)` and `assume(A,'clear')`	Instead of `sym`, use `assume` to set and clear assumptions on variables in the workspace.
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

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:

- “Find Almost Integers with High-Precision Arithmetic”. 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.
- “Matrix Rotations and Transformations”. 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 `sym` and `vpa` are restricted to numbers and valid variable names.	Still runs	When creating symbolic expressions, first create symbolic variables, and then use operations on them. For example, use:
		• `syms x; x + 1`
		instead of
		`sym('x + 1')`
		• `exp(sym(pi))`
		instead of
		`sym('exp(pi)')`
		• `syms f(var1,...varN)`
		instead of
		`f(var1,...varN) = sym('f(var1,...varN)')`
		• `vpa((1 + sqrt(sym(5)))/2)`
		instead of
		`vpa('(1 + sqrt(5))/2')`
			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.

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 of character vector, 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
Values `All` and `None` of `IgnoreAnalyticConstraints`	Warns	`true` and `false`	Replace all instances of `'IgnoreAnalyticConstraints','All'` with `'IgnoreAnalyticConstraints',true`. Replace all instances of `'IgnoreAnalyticConstraints','None'` with `'IgnoreAnalyticConstraints',false`.
`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 Special Functions for the list of available special functions.
`mfunlist`	Warns	See Special Functions.	See Special Functions for the list of available special functions.
`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

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.
- "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`symdiffdemo` 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
`findsym`	Still runs	`symvar`	Replace all instances of `findsym` with `symvar`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Values `All` and `None` of `IgnoreAnalyticConstraints`	Still runs	`true` and `false`	Replace all instances of `'IgnoreAnalyticConstraints','All'` with `'IgnoreAnalyticConstraints',true`. Replace all instances of `'IgnoreAnalyticConstraints','None'` with `'IgnoreAnalyticConstraints',false`.
`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 Special Functions for the list of available special functions.
`mfunlist`	Warns	See Special Functions.	See Special Functions for the list of available special functions.
`setVar(nb, 'MuPADvar')`	Warns	Three-argument version `setVar(nb, 'MuPADvar', MATLABexpr)`	Replace all instances of `setVar(nb,'MuPADvar')` with `setVar(nb, 'MuPADvar', MATLABexpr)`
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 of character vectors, create symbolic variables using `syms`, and then pass them as input arguments separated by commas, or as a vector of input arguments.
`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'))`
`simple`	Errors	`simplify`	Replace all instances of `simple(S)` with `simplify(S)`. There is no replacement for `[r,how] = simple(S)`.
In previous releases, `in(x, type)` in some cases returned logical `1` if `x` belonged to `type` and `0` otherwise.	Returns a symbolic expression of the form `in(x, type)`	`isAlways`	To obtain the same results as in previous releases, wrap such expressions in `isAlways`. For example, use `isAlways(in(sym(5), 'integer'))`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
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	`isAlways`	To obtain the same results as in previous releases, wrap equations in `isAlways`. For example, use `isAlways(A < B)`.

R2014b

MATLAB solve function returning parameters and conditions in solutions

The 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.

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, `singIm(z) = sign(imag(z))`. For real numbers, `signIm(z) = -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 kroneckerDelta function for computing the Kronecker delta function

The MATLAB symbolic `kroneckerDelta` 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: `mod(a,b) = a - b*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 <= r < |b|`, and `q = a 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 `modp(a,b) = modp(u*w,b)`. Here, `w` is an inverse of `v mod b`, that is, `v*w ≡ 1 (mod b)`.
- If `a` and `b` are integers, then `mods(a,b)` is an integer `r`, such that `a = qb + r` and `-|b|/2 < r <= |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 `mods(a,b) = mods(u*w,b)`. Here, `w` is an inverse of `v mod b`, that is, `v*w ≡ 1 (mod b)`.

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

- `modp(a,b) = a - b*floor(a/b)`
- `mods(a,b) = a - b*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 `mod(a,b) = 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
`mfun`	Still runs	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 Special Functions for the list of available special functions.
`mfunlist`	Still runs	See Special Functions.	See Special Functions for the list of available special functions.
`setVar(nb, 'MuPADvar')`	Warns	Three-argument version `setVar(nb, 'MuPADvar', MATLABexpr)`	Replace all instances of `setVar(nb, 'MuPADvar')` with `setVar(nb, 'MuPADvar', MATLABexpr)`
`simple`	Warns	`simplify`	Replace all instances of `simple(S)` with `simplify(S)`. There is no replacement for `[r,how] = simple(S)`.
`emlBlock`	Errors	`matlabFunctionBlock`	Replace all instances of `emlBlock` with `matlabFunctionBlock`.

R2014a

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 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 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 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, `erfc(k,x) = int(erfc(k - 1, y), y, x, inf)`.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD `linalg::curl`	Still runs	`curl`	Replace all instances of `linalg::curl` with `curl`.
MuPAD `linalg::det`	Still runs	`det`	Replace all instances of `linalg::det` with `det`.
MuPAD `linalg::divergence`	Still runs	`divergence`	Replace all instances of `linalg::divergence` with `divergence`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD linalg::grad	Still runs	gradient	Replace all instances of linalg::grad with gradient.
MuPAD linalg::gradient	Still runs	gradient	Replace all instances of linalg::gradient with gradient.
MuPAD linalg::hessian	Still runs	hessian	Replace all instances of linalg::hessian with hessian.
MuPAD linalg::jacobian	Still runs	jacobian	Replace all instances of linalg::jacobian with jacobian.
MuPAD linalg::laplacian	Still runs	laplacian	Replace all instances of linalg::laplacian with laplacian.
MuPAD linalg::potential	Still runs	potential	Replace all instances of linalg::potential with potential.
MuPAD linalg::vectorPotential	Still runs	vectorPotential	Replace all instances of linalg::vectorPotential with vectorPotential.
simple	Warns	simplify	Replace all instances of simple(S) with simplify(S). There is no replacement for [r, how] = simple(S).
emlBlock	Errors	matlabFunctionBlock	Replace all instances of emlBlock with matlabFunctionBlock.

R2013b

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 `NaN`s.

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
simple	Still runs	simplify	Replace all instances of simple(S) with simplify(S). There is no replacement for [r, how] = simple(S).
emlBlock	Warns	matlabFunctionBlock	Replace all instances of emlBlock with matlabFunctionBlock.
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::SparseMatrixF2 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

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 simple(S) with simplify(S). There is no replacement for [r, how] = simple(S).
emlBlock	Warns	matlabFunctionBlock	Replace all instances of emlBlock with matlabFunctionBlock.
MuPAD factoring functions numlib::mpqs, numlib::pollard, and numlib::ecm	Warns	ifactor	Replace all instances of numlib::mpqs, numlib::pollard, and numlib::ecm with ifactor.
MuPAD Dom::SparseMatrixF2 domain	Warns	Dom::Matrix(Dom::IntegerMod(2))	Replace all instances of Dom::SparseMatrixF2 with Dom::Matrix(Dom::IntegerMod(2)).
MuPAD userinfo	Warns	print	Use print instead of userinfo.
MuPAD setuserinfo	Warns	prog::trace or debug	Try using prog::trace or debug instead of setuserinfo.
poly	Errors	charpoly	Replace all instances of poly with charpoly.
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	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.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
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.
MuPAD transform::invfourier::addpattern	Errors	MuPAD ifourier::addpattern	Replace all instances of transform::invfourier::addpattern with ifourier::addpattern.
MuPAD transform::laplace::addpattern	Errors	MuPAD laplace::addpattern	Replace all instances of transform::laplace::addpattern with laplace::addpattern.
MuPAD transform::invlaplace::addpattern	Errors	MuPAD ilaplace::addpattern	Replace all instances of transform::invlaplace::addpattern with ilaplace::addpattern.
MuPAD transform::ztrans::addpattern	Errors	MuPAD ztrans::addpattern	Replace all instances of transform::ztrans::addpattern with ztrans::addpattern.
MuPAD transform::invztrans::addpattern	Errors	MuPAD iztrans::addpattern	Replace all instances of transform::invztrans::addpattern with iztrans::addpattern.
MuPAD prog::calledFrom	Errors	context(hold(procname))	Replace all instances of prog::calledFrom() with context(hold(procname)).
MuPAD prog::calltree	Errors	prog::trace	Use prog::trace instead of prog::calltree.
MuPAD prog::error	Errors	getlasterror	Use getlasterror instead of prog::error.
MuPAD prog::memuse	Errors	prog::trace(Mem)or bytes()	Use prog::trace(Mem) or bytes() instead of prog::memuse.
MuPAD prog::testfunc	Errors	print(Unquoted, "...")	Use print(Unquoted, "...") instead of prog::testfunc.
MuPAD prog::testmethod	Errors	prog::test(..., Method = myTestMethod)	Use prog::test(..., Method = myTestMethod) instead of prog::testmethod.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
MuPAD `prog::testnum`	Errors	Nothing	No replacement
Dynamic modules for MuPAD, including the `module`, `external`, and `Pref::unloadableModules` functions and all functions of the `module` library	Errors	Nothing	No replacement

R2012b

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 `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.

`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)`.

`diff` and `int` methods for inputs of the `char` type	Errors	`sym`	Use the `sym` method instead.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`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`.
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

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 `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

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
`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

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

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

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 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

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
`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

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 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

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 `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.

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

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*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

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

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.

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+

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*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.
- `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 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
`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

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

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.