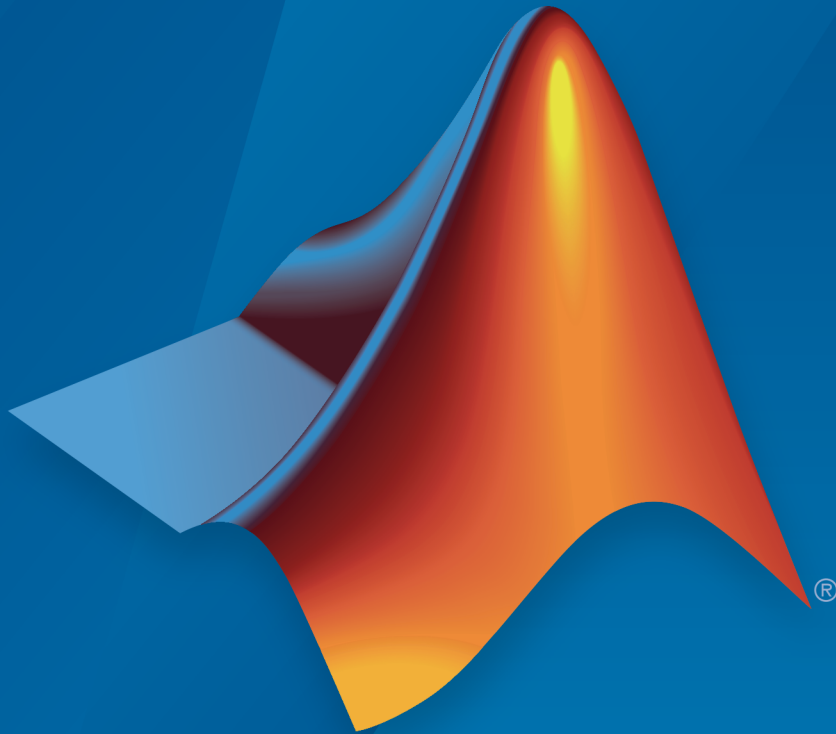


Partial Differential Equation Toolbox™ Release Notes



MATLAB®

How to Contact MathWorks



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



Phone: 508-647-7000



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

Partial Differential Equation Toolbox™ Release Notes

© COPYRIGHT 2012–2017 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

R2017a

Thermal Analysis: Solve conduction-dominant heat transfer problems with convection and radiation occurring at boundaries	1-2
3-D Multidomain Geometry: Create geometries consisting of stacked or nested cuboids, spheres, or cylinders	1-3
Functionality being removed or changed	1-3

R2016b

Flux of PDE Solution: Evaluate tensor product of c-coefficient and gradient of PDE solution for 2-D and 3-D problems	2-2
Boundary Conditions: Set an individual boundary condition for each equation in a PDE system, query boundary condition assignments	2-2
Nodal Initial Conditions: Set nodal initial conditions using the result of a previous simulation	2-3
Geometric Initial Conditions: Set initial conditions on vertices, edges, faces, and cells	2-3
Mesh and Solution Plots: Set transparency, display node and element labels	2-3

3-D Geometry Import from STL Files: Improved quality of the resulting geometry	2-4
Functionality being removed or changed	2-4

R2016a

PDE Solvers: Use <code>solvepde</code> and <code>solvepdeeig</code> functions to solve PDEs and PDE eigenvalue problems	3-2
PDE Coefficients: Specify equation coefficients as a property of PDEModel	3-2
Initial Conditions: Set initial conditions or initial guess as a property of PDEModel	3-2
Quadratic Elements for 2-D Mesh: Generate 2-D mesh using quadratic triangular elements	3-2
Gradient of PDE Solution: Evaluate solution gradient at arbitrary 2-D or 3-D points	3-3
Finite Element Matrices: Use <code>assembleFEMatrices</code> to assemble finite element matrices	3-3
PDE Results for Plotting and Postprocessing: New result objects depend on the type of PDE	3-3
Functionality being removed or changed	3-4

R2015b

3-D geometry creation from a finite element mesh using the <code>geometryFromMesh</code> function	4-2
--	-----

Data structure to represent solutions using the createPDEResults function	4-2
Functionality being removed or changed	4-2

R2015a

3-D finite element analysis	5-2
Equation coefficients and boundary conditions for 3-D problems	5-2
Elliptic, parabolic, hyperbolic, nonlinear, eigenvalue solvers for 3-D problems	5-2
3-D geometry import from STL files	5-3
3-D unstructured meshing using tetrahedra	5-3
Plot function to inspect 3-D solutions	5-3
Featured examples with 3-D geometry	5-3
pdebound and pdegeom reference pages removed	5-3

R2014b

Functions for modular definition of boundary conditions ..	6-2
pdeInterpolant object for solution interpolation	6-2

R2014a

Damping option for hyperbolic solver 7-2

R2013b

Display option in hyperbolic and parabolic solvers 8-2

Eigenvalue example 8-2

R2013a

**Performance and robustness enhancements in meshing
algorithm** 9-2

New example 9-2

R2012b

**Coefficients of parabolic and hyperbolic PDEs that can be
functions of the solution and its gradient** 10-2

Graphics export from pdetool 10-2

pdegplot labels edges and subdomains 10-2

New examples 10-2

pdesmech shear strain calculation change 10-3

R2017a

Version: 2.4

New Features

Bug Fixes

Compatibility Considerations

Thermal Analysis: Solve conduction-dominant heat transfer problems with convection and radiation occurring at boundaries

A new programmatic workflow lets you set up, solve, and analyze conduction-dominant heat transfer problems using natural steps and familiar domain-specific language. When setting up a heat transfer problem, you can:

- Create a special model container for a steady-state or transient thermal model. See `createpde`.
- Assign thermal properties of the material, such as thermal conductivity, specific heat, and mass density. See `thermalProperties`.
- Specify internal heat sources of the geometry. See `internalHeatSource`.
- Set temperatures on boundaries, specify insulated boundaries, heat fluxes through boundaries, convection coefficients, radiation emissivity coefficients, and ambient temperature. See `thermalBC`.
- Set initial temperature or initial guess for temperature. See `thermalIC`.

The `solve` function solves steady-state and transient thermal models and returns temperatures and temperature gradients at nodal locations of the thermal model mesh. You can analyze the resulting solution further by using one of these functions:

- `interpolateTemperature` interpolates resulting temperatures to arbitrary spatial locations.
- `evaluateTemperatureGradient` evaluates temperature gradient for a transient thermal solution at arbitrary spatial locations.
- `evaluateHeatFlux` evaluates heat flux of a thermal solution at nodal or arbitrary spatial locations.
- `evaluateHeatRate` evaluates integrated heat flow rate normal to specified boundary.

The `pdeplot` and `pdeplot3D` functions let you visualize a thermal model solution by plotting the resulting temperatures, and temperature gradients, and mesh.

There is a new featured example showing heat transfer in a transient 3-D model, “Heat Conduction in a Spherical Multidomain Geometry with Nonuniform Heat Flux”.

For more details about the new workflow, see “Heat Transfer”.

3-D Multidomain Geometry: Create geometries consisting of stacked or nested cuboids, spheres, or cylinders

The `multicuboid`, `multicylinder`, and `multisphere` functions create 3-D geometries formed by several cubic, cylindrical, and spherical cells, respectively. With these functions, you can create stacked or nested geometries. You also can create geometries where some cells are empty, for example, hollow cylinders, cubes, or spheres.

All cells in a geometry must be of the same type: either cuboids, or cylinders, or spheres. These functions do not combine cells of different types in one geometry.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>pdemdlcv</code>	Errors	No replacement	Remove all instances of <code>pdemdlcv</code> .
Function handle for specifying nonconstant boundary conditions and coefficients of the form <code>@f(problem, region, state)</code> .	Still runs	<code>@f(region, state)</code>	Use the two-argument form, <code>@f(region, state)</code> , to define a function handle for <code>applyBoundaryCondition</code> or <code>specifyCoefficients</code> .

R2016b

Version: 2.3

New Features

Bug Fixes

Compatibility Considerations

Flux of PDE Solution: Evaluate tensor product of c-coefficient and gradient of PDE solution for 2-D and 3-D problems

The `evaluateCGradient` function computes the tensor product of `c`-coefficient and gradients of the PDE solution, $\mathbf{c} \otimes \nabla \mathbf{u}$, at nodal or arbitrary locations in 2-D or 3-D geometry.

Boundary Conditions: Set an individual boundary condition for each equation in a PDE system, query boundary condition assignments

The `applyBoundaryCondition` function can set an individual boundary condition for each equation in a system of PDEs in one function call. For example, in a system of two PDEs, one `applyBoundaryConditions` call can set the Dirichlet boundary condition for the first equation, and the Neumann boundary condition for the second equation.

The function uses clear precedence rules to decide which condition to use for every geometric region.

The `findBoundaryConditions` function returns the currently used boundary condition assignment for a geometric region.

Compatibility Considerations

The new syntax requires explicit specification of the boundary condition type as `'dirichlet'`, `'neumann'`, or `'mixed'`. For example, in previous releases you used the following syntaxes to apply the Dirichlet boundary condition on edge 1 and Neumann boundary condition on face 2.

```
applyBoundaryCondition(model,'Edge',1,'u',0);  
applyBoundaryCondition(model,'Face',2,'g',0,'q',0);
```

Update them to specify the boundary condition types.

```
applyBoundaryCondition(model,'dirichlet','Edge',1,'u',0);  
applyBoundaryCondition(model,'neumann','Face',2,'g',0,'q',0);
```

Previously, to set the Dirichlet boundary condition for one equation in a PDE system and the Neumann boundary condition for another, you used two function calls to `applyBoundaryCondition`.

```
applyBoundaryCondition(model,'edge',4,'u',0,'EquationIndex',1);
```

```
applyBoundaryCondition(model, 'edge', 4, 'g', [0, -1]);
```

Now, set both boundary conditions in one function call by specifying the boundary condition type as 'mixed'.

```
applyBoundaryCondition(model, 'mixed', 'edge', 4, 'u', 0, 'EquationIndex', 1, 'g', [0, -1]);
```

Nodal Initial Conditions: Set nodal initial conditions using the result of a previous simulation

The `setInitialConditions` function sets initial conditions at the mesh nodes by using the solution from a previous analysis on the same geometry and mesh.

Geometric Initial Conditions: Set initial conditions on vertices, edges, faces, and cells

The `setInitialConditions` function sets initial conditions on faces, edges, and vertices for both 2-D and 3-D geometries, and on cells for 3-D geometry. You can set initial conditions on particular vertices, edges, faces, and cells or set global initial conditions on all vertices, edges, faces, or cells in the geometry.

Mesh and Solution Plots: Set transparency, display node and element labels

The `FaceAlpha` argument of `pdegplot`, `pdemesh`, `pdeplot`, and `pdeplot3D` lets you set the plot transparency.

The `ElementLabels` argument of `pdemesh`, `pdeplot`, and `pdeplot3D` lets you display element labels.

The `NodeLabels` argument of `pdemesh` and `pdeplot` lets you display node labels.

The `VertexLabels` and `CellLabels` arguments of `pdegplot` let you display vertex and cell labels.

Compatibility Considerations

The argument `SubdomainLabels` is no longer recommended. Use `FaceLabels` for 2-D geometries instead.

3-D Geometry Import from STL Files: Improved quality of the resulting geometry

When importing an STL geometry for a 3-D problem, `importGeometry` can recognize and reconstruct more geometric vertices, edges, and faces of the original CAD geometry in some instances. In these cases, the resulting geometry is a closer match to the original CAD geometry.

Compatibility Considerations

Detailed geometries can now contain more faces and edges than in previous releases. As a result, in rare instances, the new faces and edges can cause renumbering of the existing ones. If your code imports an STL geometry, visually check the geometry to ensure that you are assigning boundary and initial conditions to the intended regions.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
<code>pdemd1cv</code>	Warns	No replacement	Remove all instances of <code>pdemd1cv</code> .
<code>pde</code>	Errors	<code>createpde</code>	Use <code>createpde</code> to create a <code>PDEModel</code> that holds the PDE analysis data. The <code>pde</code> class was a value class. The replacement <code>PDEModel</code> class is a handle class.
<code>pdeGeometryFromEdg</code>	Errors	<code>geometryFromEdges</code>	Use <code>geometryFromEdges</code> instead. Although <code>pdeBoundaryConditions</code> still runs with a warning, its returned type has changed from a <code>pdeGeometry</code> object to an <code>AnalyticGeometry</code> object.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
			The <code>pdeGeometry</code> class was a value class. The replacement <code>AnalyticGeometry</code> class is a handle class.
<code>pdeBoundaryConditions</code>	Errors	<code>applyBoundaryConditions</code>	<p>Replace all instances of <code>pdeBoundaryConditions</code> (<code>ApplicationRegion</code>) with <code>applyBoundaryCondition(model, BCType, 'e')</code>.</p> <p>Although <code>pdeBoundaryConditions</code> still runs with a warning, its returned type has changed from a <code>pdeBoundaryConditions</code> object to a <code>1BoundaryCondition</code> object. The <code>pdeBoundaryConditions</code> class was a value class. The replacement <code>BoundaryCondition</code> class is a handle class.</p>
Function handle for specifying nonconstant boundary conditions and coefficients of the form <code>@f(problem, region, state)</code> .	Still runs	<code>@f(region, state)</code>	<p>Use <code>specifyCoefficients</code> and define a function handle that takes only two arguments:</p> <pre>@f(region, state)</pre>

R2016a

Version: 2.2

New Features

Bug Fixes

Compatibility Considerations

PDE Solvers: Use `solvepde` and `solvepdeeig` functions to solve PDEs and PDE eigenvalue problems

New solver functions for `PDEModel`:

- `solvepde` replaces `asempde`, `pdenonlin`, `hyperbolic`, and `parabolic`.
- `solvepdeeig` replaces `pdeeig`.

To use the new solvers, include PDE coefficients in your model using `specifyCoefficients`. Include initial conditions in your model using `setInitialConditions`.

The new solvers return results as one of the three new objects:

- `StationaryResults` — Returned by `solvepde` for a stationary PDE model.
- `TimeDependentResults` — Returned by `solvepde` for a time-dependent PDE model.
- `EigenResults` — Returned by `solvepdeeig`.

`StationaryResults` and `TimeDependentResults` objects contain solution gradients at the nodes.

PDE Coefficients: Specify equation coefficients as a property of `PDEModel`

The `specifyCoefficients` function specifies equation coefficients as a property of `PDEModel`.

Initial Conditions: Set initial conditions or initial guess as a property of `PDEModel`

The `setInitialConditions` function specifies initial conditions as a property of `PDEModel`.

Quadratic Elements for 2-D Mesh: Generate 2-D mesh using quadratic triangular elements

Create a quadratic mesh for 2-D problems using `generateMesh` with `GeometricOrder` set to `'quadratic'`.

Gradient of PDE Solution: Evaluate solution gradient at arbitrary 2-D or 3-D points

The `evaluateGradient` function enables you to interpolate the gradient of a `StationaryResults` or `TimeDependentResults` object at arbitrary points in the geometry.

Finite Element Matrices: Use `assembleFEMatrices` to assemble finite element matrices

The `assembleFEMatrices` function assembles finite element matrices for independent factoring and solution using linear algebra methods. It replaces `asempde`, `asema`, and `assemb` for matrix assembly.

PDE Results for Plotting and Postprocessing: New result objects depend on the type of PDE

The `createPDEResults` function returns results as one of the three new objects, depending on the type of the PDE problem.

- A `StationaryResults` object for a stationary PDE model. `StationaryResults` contains the solution of PDE and its gradients at the nodal locations.
- A `TimeDependentResults` object for a time-dependent PDE model. `TimeDependentResults` contains the solution of PDE and its gradients at the nodal locations.
- A `EigenResults` object for an eigenvalue problem.

Compatibility Considerations

`createPDEResults` no longer creates an object of type `PDEResults`.

The syntax of `createPDEResults` has changed to accommodate creating the new result types for time-dependent and eigenvalue problems.

- To create the `TimeDependentResults` object for a time-dependent problem, use the syntax `createPDEResults(pdem,u,utimes,'time-dependent')`, where `utimes` is a vector of solution times.
- To create the `EigenResults` object for an eigenvalue problem, use the syntax `createPDEResults(pdem,eigenvectors,eigenvalues,'eigen')`.

EigenResults has different property names than PDEResults. Update any eigenvalue scripts that use PDEResults property names.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
wbound	Still runs	No replacement	New features and the recommended workflow are not compatible with this function. For the recommended workflow, see Solve Problems Using PDEModel Objects.
pdeadgsc and pdeadworst	Still runs	No replacement	New features and the recommended workflow are not compatible with these functions. For the recommended workflow, see Solve Problems Using PDEModel Objects.
asempde, assema, assemb, hyperbolic, parabolic, pdenonlin	Still runs	solvepde and assembleFEMatrices	To solve PDE problems, use solvepde. To obtain finite element matrices, use assembleFEMatrices. For the recommended workflow, see Solve Problems Using PDEModel Objects.
pdeeig and sptarn	Still runs	solvepdeeig	To solve PDE eigenvalue problems, use solvepdeeig. For the recommended workflow, see Solve Problems Using PDEModel Objects.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
poimesh, poiasma, poicalc, poiindex, poisolv	Still runs	solvepde	To solve Poisson's equations, use solvepde. For details, see Solve Problems Using PDEModel Objects.
pdejmps	Still runs	No replacement	New features and the recommended workflow are not compatible with this function. For the recommended workflow, see Solve Problems Using PDEModel Objects.
pdesmech	Still runs	PDE app	Use the PDE app instead of pdesmech.
dst and idst	Still runs	No replacement	Remove all instances of dst and idst.
tri2grid and pdeintrp	Still runs	interpolateSolution	Use the interpolation function interpolateSolution provided by StationaryResults, TimeDependentResults and EigenResults.
pdeprtni	Still runs	No replacement	NodalSolution is a property of StationaryResults and TimeDependentResults. Eigenvectors is the corresponding property of EigenResults.
pdemdlcv	Still runs	No replacement	Remove all instances of pdemdlcv.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
pde	Warns	createpde	<p>Use createpde to create a PDEModel that holds the PDE analysis data.</p> <p>The pde class was a value class. The replacement PDEModel class is a handle class.</p>
pdeGeometryFromEdges	Warns	geometryFromEdges	<p>Use geometryFromEdges instead.</p> <p>Although pdeBoundaryConditions still runs with a warning, its returned type has changed from a pdeGeometry object to an AnalyticGeometry object. The pdeGeometry class was a value class. The replacement AnalyticGeometry class is a handle class.</p>
pdeBoundaryConditions	Warns	applyBoundaryConditions	<p>Replace all instances of pdeBoundaryConditions(ApplicationRegion) with applyBoundaryCondition(model, 'edge', Edges).</p> <p>Although pdeBoundaryConditions still runs with a warning, its returned type has changed from a pdeBoundaryConditions object to a BoundaryCondition object. The pdeBoundaryConditions class was a value class. The replacement BoundaryCondition class is a handle class.</p>

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
Loading a <code>pdeBoundaryCondition</code> object from an R2014b MAT file.	Errors	<code>applyBoundaryCondition</code>	Recreate the Boundary Conditions using <code>applyBoundaryCondition</code> .
Function handle for specifying nonconstant boundary conditions and coefficients of the form <code>@f(problem, region, state)</code> .	Still runs	<code>@f(region, state)</code>	Use <code>specifyCoefficients</code> and define a function handle that takes only two arguments: <code>@f(region, state)</code>

R2015b

Version: 2.1

New Features

Bug Fixes

Compatibility Considerations

3-D geometry creation from a finite element mesh using the `geometryFromMesh` function

The `geometryFromMesh` function creates 3-D geometry from a finite element mesh, or from a triangulated surface mesh. For details, see the function reference page or [Create and View 3-D Geometry](#).

Data structure to represent solutions using the `createPDEResults` function

The `createPDEResults` function converts a PDE solution into a `PDEResults` object. The `PDEResults` object allows you to interpolate the solution using `interpolateSolution`. For details, see the reference pages.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>pdesmech</code>	Still runs	PDE app	Use the PDE app instead of <code>pdesmech</code>

R2015a

Version: 2.0

New Features

Bug Fixes

Compatibility Considerations

3-D finite element analysis

You can now solve partial differential equations with 3-D geometry. To do so, there is a new workflow that combines the geometry, mesh, and boundary conditions into a PDEModel object. You can also use this workflow for 2-D geometry. For details, see [Solve Problems Using PDEModel Objects](#).

Equation coefficients and boundary conditions for 3-D problems

To specify problem coefficients or boundary conditions in 3-D geometry, you can use strings with a syntax similar to that of a 2-D problem. There is a new way of writing functions for coefficients in 3-D geometries. For details, see [PDE Coefficients and Boundary Conditions](#).

Compatibility Considerations

To accommodate both 2-D and 3-D geometry, the format of boundary condition objects changed from that introduced in R2014b. The new object is `BoundaryCondition` Properties, and calling `pdeBoundaryConditions` now warns that it will be removed in a future release. If you saved a `pdeBoundaryConditions` object in an R2014b-format MAT file, then loading that file in R2015a can produce an error. Additionally, the syntax for specifying nonconstant boundary conditions has changed. Functions written in the previous syntax continue to work for now.

R2014b Syntax	R2015a Syntax
<code>function bcMatrix = myfun(problem,region,state)</code>	<code>function bcMatrix = myfun(region,state)</code>

For details, see [Changes to Boundary Conditions Object From R2014b](#).

Elliptic, parabolic, hyperbolic, nonlinear, eigenvalue solvers for 3-D problems

The main toolbox solvers now support problems with 3-D geometry. For a listing of functions that do or do not support 3-D geometry, see [Functions That Support 3-D Geometry](#). Solvers take a `model` argument instead of the previous `b`, `p`, `e`, `t` arguments. For details, see the function reference pages.

3-D geometry import from STL files

Import the geometry for a 3-D problem in the STL file format using the `importGeometry` function. For details, see [Create and View 3-D Geometry](#).

3-D unstructured meshing using tetrahedra

Create finite element meshes using the `generateMesh` function. For 3-D geometry, the meshes consist of tetrahedra. See [Mesh Data for \[p,e,t\] Triples: 3-D](#).

Plot function to inspect 3-D solutions

The `pdeplot3D` function plots solutions on the boundaries of 3-D geometry. For details, see [Plot 3-D Solutions](#).

Featured examples with 3-D geometry

There are two new featured examples related to linear elasticity that have 3-D geometry:

- [Deflection Analysis of a Bracket](#)
- [Vibration of a Square Plate](#)

There is also a new example of plotting slices through a 3-D solution: [Contour Slices Through a 3-D Solution](#).

To run the examples at the MATLAB[®] command line:

```
echodemo StrainedBracketExample  
echodemo Eigenvaluesofa3DPlateExample  
echodemo ContourSlices3DExample
```

`pdebound` and `pdegeom` reference pages removed

The `pdebound` and `pdegeom` reference pages have been replaced by the [Boundary Conditions](#) and [2-D Geometry](#) documentation categories.

R2014b

Version: 1.5

New Features

Bug Fixes

Functions for modular definition of boundary conditions

To specify PDE boundary conditions in a modular fashion, per edge or set of edges, use a `pdeBoundaryConditions` specification. For details, see [Steps to Specify a Boundary Conditions Object](#).

`pdeInterpolant` object for solution interpolation

Interpolate a PDE solution to a set of points using `evaluate` on an interpolant. Create the interpolant using `pdeInterpolant`.

R2014a

Version: 1.4

New Features

Bug Fixes

Damping option for hyperbolic solver

You can include damping in the hyperbolic solver in matrix form. There is a new example of dynamics of a damped cantilever beam that shows how to use this feature.

R2013b

Version: 1.3

New Features

Bug Fixes

Display option in hyperbolic and parabolic solvers

You can disable the display of internal ODE solution details that the hyperbolic and parabolic solvers report. To disable the display, set the `Stats` name-value pair to `'off'`.

Eigenvalue example

There is a new example of eigenvalues of a circular membrane. View the example [here](#). To run the example at the MATLAB command line:

```
echodemo eigsExample
```

R2013a

Version: 1.2

New Features

Bug Fixes

Performance and robustness enhancements in meshing algorithm

The meshing (geometry triangulation) functions in `initmesh` and `adaptmesh` provide an enhancement option for increased meshing speed and robustness. Choose the enhanced algorithm by setting the `MesherVersion` name-value pair to `'R2013a'`. The default `MesherVersion` value of `'preR2013a'` gives the same mesh as previous toolbox versions.

The enhancement is available in `pdetool` from the **Mesh > Parameters > Mesher version** menu.

New example

There is a new example of heat distribution in a radioactive rod. View the example [here](#). To run the example at the MATLAB command line:

```
echodemo radioactiveRod
```

R2012b

Version: 1.1

New Features

Compatibility Considerations

Coefficients of parabolic and hyperbolic PDEs that can be functions of the solution and its gradient

You can now solve parabolic and hyperbolic equations whose coefficients depend on the solution u or on the gradient of u . Use the parabolic or hyperbolic commands, or solve the equations using `pdeTool`. For details, see the function reference pages.

Graphics export from `pdeTool`

You can save the current `pdeTool` figure in a variety of image formats. Save the figure using the **File > Export Image** menu. See File Menu.

`pdegplot` labels edges and subdomains

`pdegplot` now optionally labels:

- The edges in the geometry
- The subdomains in the geometry

To obtain these labels, set the `edgeLabels` or `subdomainLabels` name-value pairs to 'on'. For details, see the `pdegplot` reference page.

New examples

There is a new example of uniform pressure load on a thin plate. View the example here. To run the example at the MATLAB command line:

```
echodemo clampedSquarePlateExample
```

There is a new example of nonlinear heat transfer in a thin plate. View the example here. To run the example at the MATLAB command line:

```
echodemo heatTransferThinPlateExample
```

There is a new example of a system of coupled PDEs. View the example here. To run the example at the MATLAB command line:

```
echodemo deflectionPiezoelectricActuator
```

pdesmech shear strain calculation change

The pdesmech function now calculates shear strain according to the engineering shear strain definition. This has always been the documented behavior. However, the previous calculation was performed according to the tensor shear strain calculation, which gives half the value of the engineering shear strain.

Compatibility Considerations

pdesmech now returns shear strain values exactly twice as large as before.

