

MATLAB® Release Notes



MATLAB®



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*MATLAB® Release Notes*

© COPYRIGHT 2004-2022 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](http://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](http://www.mathworks.com/patents) for more information.

<b>Environment</b> .....	<b>1-2</b>
Themes in MATLAB Online: Change the colors of the MATLAB desktop by selecting a dark or light theme .....	<b>1-2</b>
Live Editor Colors: Change the text and background colors of live scripts and functions .....	<b>1-3</b>
Live Editor Hyperlinks: Insert hyperlinks to specific locations in separate live scripts or live functions .....	<b>1-3</b>
Live Editor Export: Export live scripts and functions programmatically using the export function .....	<b>1-3</b>
Live Editor Accessibility: Interact with output in live scripts using the keyboard .....	<b>1-3</b>
Live Editor Tasks: View and interact with tasks when code is hidden .....	<b>1-4</b>
Component Browser: Reorder children in App Designer or the Property Inspector .....	<b>1-4</b>
Editor Python Support: View and edit Python files with syntax highlighting, auto-indenting, and delimiter matching .....	<b>1-4</b>
Find and Replace Dialog Box: Search text in the Editor and Live Editor using regular expressions .....	<b>1-4</b>
Profiler: Access the Profiler from the Apps tab .....	<b>1-5</b>
Internationalization: UTF-8 system encoding on Windows platforms .....	<b>1-5</b>
Installation Settings: Configure persistent settings for MATLAB installations .....	<b>1-5</b>
Comparison Tool: Save results as HTML report .....	<b>1-5</b>
Comparison Tool: Compare folders in MATLAB Online .....	<b>1-5</b>
Functionality being removed or changed .....	<b>1-5</b>
<b>Language and Programming</b> .....	<b>1-7</b>
Class Introspection: Description and DetailedDescription properties of metaclasses contain text from code comments .....	<b>1-7</b>
Class Introspection: Access class aliases from meta.class instance .....	<b>1-7</b>
Background Pool: See futures in the background .....	<b>1-7</b>
cancelAll Method: Cancel currently queued and running futures in the background pool .....	<b>1-7</b>
Background Pool: Check the status of the background pool .....	<b>1-7</b>
pcode Function: Create P-code files with enhanced obfuscation .....	<b>1-7</b>
str2num Function: Restrict evaluation to basic math expressions .....	<b>1-8</b>
Functionality being removed or changed .....	<b>1-8</b>
<b>Data Analysis</b> .....	<b>1-10</b>
Data Cleaner App: Interactively preprocess and organize column-oriented data .....	<b>1-10</b>

allfinite, anynan, and anymissing Functions: Determine if all array elements are finite, any element is NaN, and any element is missing . . . . .	1-10
quantile, prctile, and iqr Functions: Calculate quantiles, percentiles, and interquartile range . . . . .	1-10
rms Function: Calculate root-mean-square value . . . . .	1-10
std and var Functions: Optionally return mean as a second output . . . . .	1-11
Date and Time Functions: Some Financial Toolbox functions combined with MATLAB functions . . . . .	1-11
Date and Time Functions: Some Financial Toolbox functions moved to MATLAB . . . . .	1-12
matlab.datetime.compatibility.convertDatenum Function: Convert text timestamps and serial date numbers to datetime values in a backward-compatible way . . . . .	1-13
categorical Data Type: Use a pattern object to specify category names that match a pattern . . . . .	1-13
table and timetable Data Types: Use a pattern object to specify row, variable, and property names that match a pattern . . . . .	1-14
Data Preprocessing Functions: Append transformed variables to input data using the ReplaceValues name-value argument . . . . .	1-15
Data Preprocessing Functions: Return table with logical values using the OutputFormat name-value argument . . . . .	1-15
ismissing, rmissing, and groupsummary Functions: Accept data types with no standard missing value . . . . .	1-16
Functionality being removed or changed . . . . .	1-16
<b>Data Import and Export . . . . .</b>	<b>1-18</b>
Parquet: Read Parquet file data more efficiently using rowfilter to conditionally filter rows . . . . .	1-18
Parquet: Determine and define row groups in Parquet file data . . . . .	1-18
Parquet: Convert, import, and export nested data structures . . . . .	1-18
writelines Function: Write plain text to a file . . . . .	1-18
Reading Online Data: Use web options when reading files over HTTP and HTTPS . . . . .	1-18
Opus Files: Work with Opus (.opus) audio files. . . . .	1-19
HDF5 Interface: Write datasets using dynamically loaded filters . . . . .	1-19
NetCDF Interface: Enable byte-range reading of remote datasets . . . . .	1-19
NetCDF Interface: Read and write variable length array data types (NC_VLEN) . . . . .	1-19
Scientific File Format Libraries: NetCDF library is upgraded . . . . .	1-19
Hardware Manager App: Discover and connect to your hardware from MATLAB . . . . .	1-19
TCP/IP Client Interface: Specify transfer delay options . . . . .	1-19
Functionality being removed or changed . . . . .	1-20
<b>Mathematics . . . . .</b>	<b>1-21</b>
pagemldivide, pagemrdivide, and pageinv Functions: Solve linear equations and calculate matrix inverses using pages of N-D arrays . . . . .	1-21
tensorprod Function: Calculate tensor products between two arrays . . . . .	1-21
round Function: Control tiebreak behavior . . . . .	1-21
null and orth Functions: Specify tolerance to treat singular values below a threshold as zero . . . . .	1-21
norm Function: Frobenius norm calculations support N-D arrays . . . . .	1-21
equilibrate Function: Specify output format of factorization . . . . .	1-21

rand, randi, and randn Functions: Support for complex input and RandStream object with the "like" syntax . . . . .	1-22
eps, flintmax, intmax, intmin, realmax, and realmin Functions: Use "like" syntax to return scalars based on prototype object . . . . .	1-22
qr and gsvd Functions: Option for economy-size decompositions . . . . .	1-22
Functionality being removed or changed . . . . .	1-22
<b>Graphics . . . . .</b>	<b>1-23</b>
Plotting Table Data: Create line plots by passing tables directly to plotting functions . . . . .	1-23
Data Tips: View table variable names as row labels . . . . .	1-23
Data Tips: View visual property values for scatter plots and bubble charts . . . . .	1-24
Bubble Charts and 3-D Scatter Plots: Plot multiple data sets at once . . . .	1-25
fontname and fontsize Functions: Specify the font and font size for graphics objects . . . . .	1-26
exportgraphics Function: Create animated GIF files . . . . .	1-26
Annotation Graphics Objects: Change the annotation rotation angle with the Rotation property . . . . .	1-26
Quiver Plots: Align the heads, centers, or tails of arrows with data points . . . . .	1-26
xlim, ylim, and zlim Functions: Query the axis limit method . . . . .	1-27
view Function: Change the view on multiple axes simultaneously . . . . .	1-27
rendererinfo Function: Get renderer information without specifying the axes . . . . .	1-27
linkaxes Function: Synchronize axes in all dimensions by default . . . . .	1-27
cameratoolbar Function: Syntax support for figures created with the uifigure function . . . . .	1-28
Callbacks in Live Editor: Create callbacks for figures in the Live Editor .	1-28
Figure Code: Generate code for figure interactions in MATLAB Online . .	1-28
Functionality being removed or changed . . . . .	1-29
<b>App Building . . . . .</b>	<b>1-32</b>
uistyle Function: Add icons and format text in table cells and tree nodes . . . . .	1-32
uitable Function: Rearrange columns of table UI components interactively . . . . .	1-32
focus Function: Give keyboard focus to UI components programmatically . . . . .	1-33
isInScrollView Function: Determine if a component is visible in a scrollable container . . . . .	1-33
uigridlayout Function: Resize table, list box, and image UI components to fit content . . . . .	1-33
Live Editor Tasks: Develop your own Live Editor tasks for use in live scripts and functions . . . . .	1-34
Custom UI Components: Interactively create custom UI components in App Designer . . . . .	1-34
App Designer: Modify tab focus order of components . . . . .	1-34
App Designer: Specify error handling options and navigate from error messages when debugging an app . . . . .	1-35
App Designer: Manage image files in your app with an improved workflow . . . . .	1-35
App Designer: Convert components in a grid layout manager to use pixel- based positioning . . . . .	1-35

App Designer: Use App Designer in most modern web browsers in MATLAB Online .....	1-35
App Designer: Customize design environment layout .....	1-35
Comparison Tool: Compare and merge app files in MATLAB Online .....	1-36
Functionality being removed or changed .....	1-36
<b>Performance .....</b>	<b>1-38</b>
table Data Type Indexing: Improved performance when subscripting with dot notation or multiple levels of indexing .....	1-38
Classes: Improved performance for static methods, constant property access, and package functions in scripts .....	1-39
try Block: Improved performance when statements run error-free .....	1-40
Python Data Type Conversion: Improved performance when converting between Python and MATLAB data types in out-of-process mode .....	1-41
MATLAB Engine API for Python: Improved performance with large multidimensional arrays in Python .....	1-42
Matrix multiplication: Improved performance when multiplying sparse and full matrices .....	1-43
inv Function: Improved performance when inverting large triangular matrices .....	1-44
sprand and sprandn Functions: Improved performance when generating random sparse matrices .....	1-44
fzero Function: Improved performance .....	1-45
diff Function: Improved performance with large number of elements .....	1-45
groupsummary, groupfilter, and grouptransform Functions: Improved performance with small group size .....	1-46
nufftn Function: Improved performance with nonuniform sample points or query points .....	1-46
Variables Editor and Live Editor: Improved speed of data display when scrolling .....	1-47
App Building: Improved performance when creating UI components .....	1-47
uitable Function and UI Containers: Improved performance when updating properties successively .....	1-48
UI Components: Improved performance when setting a property with an unchanged value .....	1-49
App Designer: Improved performance when loading apps with UIAxes components off the canvas .....	1-49
Plots in Apps: Improved responsiveness for event-driven updates in apps .....	1-50
Plots in Apps: Improved responsiveness of axes interactions within apps .....	1-51
Plots in Apps: Improved responsiveness of axes interactions in plots with two y-axes .....	1-52
Plots in Apps: Faster animations in apps when multiple figures are open .....	1-53
Property Inspector: Improved performance when opening for the first time .....	1-54
<b>Software Development Tools .....</b>	<b>1-55</b>
Projects: Reduce test runtime in continuous integration workflows using the dependency cache .....	1-55
Dependency Analyzer: Save dependency graph as image .....	1-55
Code Compatibility Analyzer App: Identify and address compatibility issues against current version of MATLAB .....	1-55

Unit Testing Framework: Create test classes interactively using the Current Folder browser	1-55
Unit Testing Framework: Create temporary folders that are automatically removed	1-55
Unit Testing Framework: Generate DOCX, HTML, and PDF reports after test execution	1-56
Unit Testing Framework: Debug uncaught errors in tests	1-56
Unit Testing Framework: Collect statement and function coverage metrics for your source code	1-56
Functionality being removed or changed	1-56
<b>External Language Interfaces</b>	<b>1-58</b>
C++ Interface: Array size help text for functions and methods	1-58
C Interface: Build third-party C library interface using <code>clibgen.generateLibraryDefinition</code>	1-58
C++ Interface: Support for C++ language features	1-58
C++ Interface: Publisher options	1-58
Call MATLAB from C++: Generate C++ code Interface for MATLAB Packages, Classes, and Functions	1-59
MATLAB Data Array API: <code>matlab::data::Array</code> support for row-major order	1-59
MEX Functions: UTF-8 system encoding on Windows platforms	1-59
Python: Use <code>Name=Value</code> syntax to pass keyword arguments to Python functions	1-59
Python: Convert Python list and tuple types to MATLAB types	1-59
Perl 5.34.0: MATLAB support on Windows	1-60
Compilers: Support for Microsoft Visual Studio 2022	1-60
Functionality being removed or changed	1-60

## R2021b

<b>Environment</b>	<b>2-2</b>
Editor Selection: Select and edit a rectangular area of code	2-2
Editor Display: Zoom in and out in the Editor	2-2
Editor Code: Show code suggestions and completions automatically	2-2
Editor Debugging: Diagnose problems in scripts and functions using inline debugging controls and a breadcrumb-style function call stack	2-3
Editor Refactoring: Automatically convert selected code to a function	2-4
Editor Code: Automatically complete block endings, match delimiters, and wrap comments while editing code	2-4
Editor Sections: Create sections with an improved appearance	2-4
Editor Code: Change the case of text and code	2-5
Editor Bookmarks: Maintain bookmarks after closing a file	2-5
Live Editor Controls: Set default values for sliders, drop-down lists, check boxes, and edit fields	2-5
Live Editor Animations: Export animations to movies or animated GIFs	2-6
Live Editor Figures: Interact with real MATLAB figures and resize them with improved layouts	2-6
Live Editor: Improved performance when saving live scripts or functions	2-7

Comparison Tool: Compare and merge text files with improved usability, appearance, and syntax highlighting . . . . .	2-7
Importing Preferences from Previous Releases: MATLAB checks for preferences from R2019b or newer . . . . .	2-8
Display language: MATLAB uses Windows display language settings for selecting desktop language . . . . .	2-8
Functionality being removed or changed . . . . .	2-8
<b>Language and Programming . . . . .</b>	<b>2-10</b>
cast Function: Consistent output for all syntaxes with the same data type conversion . . . . .	2-10
Run Code in the Background: Use parallel language to run code asynchronously . . . . .	2-10
Portable Parallel Code: Share parallel code and seamlessly run in parallel . . . . .	2-10
Compact Display for Classes: Customize display of information about classes when space is limited . . . . .	2-11
Class Aliasing: Create aliases for renamed classes to maintain backward compatibility . . . . .	2-11
Modular Indexing: Customize class indexing operations individually using new superclasses . . . . .	2-11
Scalar Classes: Inherit from the matlab.mixin.Scalar superclass to ensure instances behave as scalars . . . . .	2-12
startat Function: Time zone information in datetime objects now supported . . . . .	2-12
Functionality being removed or changed . . . . .	2-12
<b>Data Analysis . . . . .</b>	<b>2-14</b>
Compute by Group Live Editor Task: Interactively summarize, transform, or filter groups of data . . . . .	2-14
Normalize Data Live Editor Task: Interactively center and scale data . . . . .	2-14
Clean Missing Data Live Editor Task: Define missing values . . . . .	2-14
trenddecomp Function: Find trends in data . . . . .	2-14
min and max Functions: Specify the comparison method for determining minimum and maximum values . . . . .	2-14
unique_tol Function: Options to control element selection and preserve range of data . . . . .	2-14
Data Preprocessing Functions: Specify table variable as sample points vector . . . . .	2-14
dateshift Function: Shift to next occurrence of weekday or weekend day . . . . .	2-15
isbetween Function: Support for open, closed, and half open intervals . . . . .	2-15
isregular Function: Support for datetime and duration data types . . . . .	2-15
istabular Function: Determine if input is a table or timetable . . . . .	2-15
retime and synchronize Functions: Median and mode methods supported . . . . .	2-15
timeofday Function: Return the date as the second output argument . . . . .	2-15
timeseries2timetable Function: Convert timeseries objects to timetables . . . . .	2-16
Functionality being removed or changed . . . . .	2-16
<b>Data Import and Export . . . . .</b>	<b>2-17</b>
sftp Function: Connect to SFTP servers . . . . .	2-17



Datastores: Specify FileSet objects as data locations for some datastores .....	2-17
Table Import: Read tables from HTML and Microsoft Word documents . .	2-17
HDF5 Interface: Use new functionality in support of HDF5 1.10.7 . . . . .	2-17
NetCDF Interface: Read and write NC_STRING data . . . . .	2-18
Scientific File Format Libraries: HDF5 and NetCDF libraries are upgraded .....	2-18
Audio, Video, and Image I/O Functions: Run functions in a thread-based environment . . . . .	2-18
Image File Format Libraries: LibTIFF library upgraded to version 4.2.0 .	2-18
New Serial Explorer and TCP/IP Explorer apps . . . . .	2-18
Functionality being removed or changed . . . . .	2-19
<b>Mathematics</b> . . . . .	<b>2-20</b>
ode78 and ode89 Functions: High-order Runge-Kutta solvers for ordinary differential equations . . . . .	2-20
pagesvd Function: Perform singular value decomposition on pages of N-D arrays . . . . .	2-20
svd Function: Option to control output format of singular values . . . . .	2-20
mpower Function: Improved algorithm for defective matrices . . . . .	2-20
Functionality being removed or changed . . . . .	2-20
<b>Graphics</b> . . . . .	<b>2-21</b>
Plotting Table Data: Create scatter plots, bubble charts, and swarm charts by passing tables directly to plotting functions . . . . .	2-21
Axes Ticks and Colors: Control the appearance of axis tick marks and tick label colors . . . . .	2-22
Create Plot Live Task: Add additional visualizations to generated plots . .	2-23
Create Plot Live Task: Control chart input syntax using configuration drop- down . . . . .	2-24
exportgraphics Function: Capture and append graphics to existing PDFs .....	2-24
stackedplot Function: Support for semilog y-axes . . . . .	2-24
Text Objects: Use editInteractions in the Interactions property to click or tap on text to edit . . . . .	2-24
dataTipTextRow Function: Customize data tip content using data properties, such as UserData . . . . .	2-24
MATLAB Online™ Accessibility: Use a screen reader to interact with figures .....	2-25
Functionality being removed or changed . . . . .	2-25
<b>App Building</b> . . . . .	<b>2-26</b>
UIAlert, uiconfirm, and uiprogressdlg Functions: Mark up text and display equations in dialog boxes . . . . .	2-26
addStyle Function: Add styles to nodes and levels in a tree UI component .....	2-26
uitable Function: Set and query table selections programmatically and control table selection options . . . . .	2-26
uitextarea Function: Program apps to respond while a user is typing in a text area component . . . . .	2-26
Run Code in the Background: Use parallel language to create more responsive apps . . . . .	2-26
App Designer: Debug code in Code View . . . . .	2-27

App Designer: Efficiently manage your app code with tools and shortcuts from Live Editor . . . . .	2-27
App Designer: Interactively modify canvas zoom level and fit canvas to view . . . . .	2-29
App Designer: Convert between similar UI components . . . . .	2-29
App Designer: Add help text for your app . . . . .	2-30
App Designer: Remove auto-reflow behavior from an app with auto-reflow . . . . .	2-30
Deployed Web Apps: Deploy web apps directly to the MATLAB Web App Server from within App Designer . . . . .	2-30
App Testing Framework: Perform press gestures on axes and UI axes with different selection types . . . . .	2-30
App Testing Framework: Perform drag gestures on axes and figures with different selection types . . . . .	2-31
App Testing Framework: Use any units of measurement in gestures at the center of components . . . . .	2-31
Functionality being removed or changed . . . . .	2-31
<b>Performance . . . . .</b>	<b>2-33</b>
table Data Type Indexing: Improved performance when assigning elements by subscripting with curly braces . . . . .	2-33
qrinsert and qrdelete Functions: Improved performance modifying QR factorizations . . . . .	2-33
Titles and Labels in Plots: Improved performance when creating and querying titles or labels in a loop . . . . .	2-34
Plot Interactions: Improved performance for rendering data tips and rotating scatter plots of large data sets . . . . .	2-35
Plots in Apps: Improved performance for creating plots . . . . .	2-35
App Designer: Improved performance when opening Start Page and loading apps . . . . .	2-36
App Designer: Improved performance when saving apps . . . . .	2-36
Comparison Tool: Improved performance when loading and saving MLAPP files . . . . .	2-37
uigridlayout Function: Improved performance when adding components spanning multiple columns with 'fit' width . . . . .	2-37
uigridlayout Function: Improved resizing performance when wrapping text in resizable columns . . . . .	2-38
Live Editor: Improved performance when saving live scripts or functions . . . . .	2-39
Data Processing Dialog Boxes: Improved resizing performance . . . . .	2-39
Figure Interactions: Improved performance when using built-in axes interactions . . . . .	2-39
UI Figures: Improved performance when displaying axes toolbar . . . . .	2-40
UI Figures: Improved performance when interacting with linked axes . . . . .	2-40
<b>Software Development Tools . . . . .</b>	<b>2-41</b>
Projects: Collaborate using projects in MATLAB Online . . . . .	2-41
Source Control: Work with files under Git in MATLAB Online . . . . .	2-41
Unit Testing Framework: Use the TestCase class template to create tests more quickly and accurately . . . . .	2-41
Unit Testing Framework: Run live-function-based tests interactively in MATLAB Online . . . . .	2-41
App Testing Framework: Perform press gestures on axes and UI axes with different selection types . . . . .	2-41

App Testing Framework: Perform drag gestures on axes and figures with different selection types . . . . .	2-42
App Testing Framework: Use any units of measurement in gestures at the center of components . . . . .	2-42
Functionality being removed or changed . . . . .	2-42
<b>External Language Interfaces . . . . .</b>	<b>2-44</b>
C++ interface: Support for C++ language features . . . . .	2-44
C++ interface: Publisher options . . . . .	2-44
Java interface: Specify JRE path for MATLAB . . . . .	2-45
Java: Call into MATLAB from a Java program called by MATLAB . . . . .	2-45
Python interface: Run Python commands and scripts from MATLAB . . . . .	2-45
Python: Support for complex multidimensional arrays . . . . .	2-45
Python: Version 3.9 support . . . . .	2-46
WSDL Web Services Documents: Apache CXF version 3.4.2 support . . . . .	2-46
Perl 5.32.1: MATLAB support on Windows . . . . .	2-46
Functionality being removed or changed . . . . .	2-47
<b>Hardware Support . . . . .</b>	<b>2-49</b>
Connect and Control Arduino board using the Arduino Explorer App . . . . .	2-49
Read data from APDS9960 sensor connected to the Arduino hardware . . . . .	2-49
Support for CAN shields on Raspberry Pi Hardware . . . . .	2-49

## R2021a

<b>Environment . . . . .</b>	<b>3-2</b>
Live Editor Controls: Create dynamic controls in live scripts by linking variables to drop-down items and slider values . . . . .	3-2
Live Editor Fonts: Change the name, style, size, and color of fonts programmatically using settings . . . . .	3-3
Live Editor Display: Specify where to display output by default . . . . .	3-3
Live Editor Functions: Run live functions interactively using the Run button in MATLAB Online . . . . .	3-4
Live Editor Bookmarks: Navigate quickly between lines . . . . .	3-4
Live Editor Animation Playback Controls: Interactive interface to control animations . . . . .	3-4
Live Editor Performance: Improved performance when saving large live scripts or functions . . . . .	3-4
Help Browser: View web documentation by default . . . . .	3-5
Documentation: View MATLAB documentation in French, Italian, and German . . . . .	3-5
MATLAB Drive: Get the location of your MATLAB Drive root folder programmatically . . . . .	3-5
Functionality being removed or changed . . . . .	3-5
<b>Language and Programming . . . . .</b>	<b>3-6</b>
Name=Value Syntax: Use name=value syntax for passing name-value arguments . . . . .	3-6

Retrieving Display Format: format function can get and set display format .....	3-6
Capturing disp Output: Use the formattedDisplayText function to store disp output as a string .....	3-7
Virtual File Storage: mkdir and rmdir will now be able to create and remove files from VFS directories .....	3-7
Function Argument Validation: Debugger and profiler is now supported .....	3-7
Class Diagram Viewer: Create graphical class diagrams to explore class details and share designs .....	3-7
Enumeration Comparisons: Use isequal to compare enumeration members with text data types .....	3-7
eval function: Context checking to resolve identifiers .....	3-8
Functionality being removed or changed .....	3-9
<b>Data Analysis</b> .....	<b>3-11</b>
Data Preprocessing Live Editor Tasks: Operate on multiple table variables and specify output format for table input .....	3-11
Clean Outlier Data Live Editor Task: Visualize results with a histogram .	3-11
fillmissing Function: Specify custom fill method .....	3-11
normalize Function: Normalize multiple data sets with same parameters .....	3-11
groupcounts Function: Display percentages of group counts .....	3-12
ts2timetable Function: Convert timeseries objects to timetables .....	3-12
table and timetable Functions: Specify dimension names using the 'DimensionNames' name-value argument .....	3-12
Functionality being removed or changed .....	3-12
<b>Data Import and Export</b> .....	<b>3-14</b>
XML Files: Read, write, and import XML files using readtable, readtimetable, and other functions .....	3-14
MATLAB API for Advanced XML Processing: Create, read, write, transform, and query XML .....	3-14
XML Files: Register XML namespace prefixes for evaluating XPath expressions using readtable,readstruct, and other functions .....	3-14
Low-level file I/O functions and remote data: Perform read and write operations on remotely stored files .....	3-15
save and load functions and remote data: Save, load, and append data to remotely stored v7.3 MAT-files .....	3-15
Reading Online Data: Read files over HTTP and HTTPS using readtable, audioread, and other reading functions .....	3-15
Parquet Data Format: Use categorical data in parquet data format .....	3-16
Datastores: Read all data from a datastore using parallel processing ...	3-16
Data Compression Functions: Improved functionality in zip/unzip and tar/ untar .....	3-16
imfinfo function: Get information about all Adobe Digital Negative (DNG) file tags .....	3-16
jsonencode: Add indentation to JSON text .....	3-16
Functionality being removed or changed .....	3-17
<b>Mathematics</b> .....	<b>3-19</b>
Graph Algorithms: Compute all paths, all cycles, and cycle basis .....	3-19

griddedInterpolant Object: Use multivalued interpolation to interpolate multiple data sets simultaneously . . . . .	3-19
eig Function: Improved algorithm for skew-Hermitian matrices . . . . .	3-19
cdf2rdf Function: Improved algorithm for all inputs . . . . .	3-19
Functionality being removed or changed . . . . .	3-19
<b>Graphics . . . . .</b>	<b>3-21</b>
Create Plot Live Editor Task: Create plots interactively and generate code . . . . .	3-21
bubblecloud Function: Visualize part-to-whole relationships . . . . .	3-21
tiledlayout Function: Control the tile indexing scheme . . . . .	3-21
PolarAxes Objects: Use the CurrentPoint property or call ginput to get the cursor location within polar axes . . . . .	3-22
Scatter Plots and Constant Lines: Create multiple scatter plots or constant lines at once . . . . .	3-22
Axis Limits: Define LimitsChangedFcn callback that executes when the limits of an axis change . . . . .	3-22
Axis Limits: Control axis limits . . . . .	3-22
exportgraphics and copygraphics Functions: Specify RGB, CMYK, or grayscale output . . . . .	3-23
colororder Function: Control colors in stacked plots . . . . .	3-23
Tick Labels: Automatically rotate tick labels . . . . .	3-23
patch and errorbar Functions: Expanded data type support . . . . .	3-24
Geographic Plots: Access basemaps using additional proxy server authentication types . . . . .	3-24
Functionality being removed or changed . . . . .	3-24
<b>App Building . . . . .</b>	<b>3-31</b>
uihyperlink Function: Add and configure clickable links in apps and on the App Designer canvas . . . . .	3-31
uitree Function: Add and configure check box trees in apps and on the App Designer canvas . . . . .	3-31
Interpreter Property: Style text and display equations in labels with HTML and LaTeX markup . . . . .	3-31
WindowState Property: Create UI figures that remain in the foreground . . . . .	3-31
scroll Function: Scroll to a location within a table UI component programmatically . . . . .	3-31
UI Component Accessibility: Select ListBox items, Table cells, ColorPicker colors, and DatePicker menus using the keyboard . . . . .	3-32
App Designer: Use custom UI components in App Designer . . . . .	3-32
App Designer: Zoom and pan in the canvas, and zoom in the Code View editor . . . . .	3-33
App Designer: Control color and tab settings in Code View using MATLAB preferences . . . . .	3-33
App Designer: Customize split-screen layouts in the App Designer editor . . . . .	3-33
App Testing Framework: Perform gestures on panels and tables . . . . .	3-33
App Testing Framework: Close alert dialog box in front of figure window . . . . .	3-34
Web Apps and Standalone Applications: Datatips supported in graphics . . . . .	3-34
Functionality Being Removed or Changed . . . . .	3-34
<b>Performance . . . . .</b>	<b>3-36</b>

Sparse Matrix Multiplication: Improved performance multiplying large sparse matrices	3-36
Sparse Linear Systems: Improved performance solving sparse linear systems $A \cdot X = B$ with multicolumn B	3-36
vecnorm Function: Improved performance operating on data with multiple columns	3-37
ismember Function: Improved performance for cell inputs	3-37
unique Function: Improved performance for numeric, logical, char, and cell inputs	3-38
Graph Functions: Improved performance modifying node and edge lists	3-39
Axes Toolbar: Appears without delay when axes are ready	3-39
Rearranging UI Components: Improved performance when rearranging UI components in a UI figure	3-39
UI Figure Interactions: Faster responses to scroll, pointer movement, and resize interactions in UI figures	3-40
Plots in Apps: Improved performance for polar plots, volume visualizations, plots with more than 16 axes, and older systems	3-41
Plots in Apps: Improved performance for plots with large numbers of markers	3-42
Live Editor: Improved performance when saving large live scripts or functions	3-42
<b>Software Development Tools</b>	<b>3-44</b>
Projects: List all referenced projects of the current project	3-44
Projects: List impacted project files	3-44
Dependency Analyzer: Find required add-ons	3-44
Unit Testing Framework: Create test runners using alternative syntax	3-44
Unit Testing Framework: Initialize parameterization properties at suite creation time	3-44
Unit Testing Framework: Run tests in parallel on thread-based pool	3-44
Unit Testing Framework: Run tests in MATLAB Online interactively	3-45
App Testing Framework: Perform gestures on panels and tables	3-45
App Testing Framework: Close alert dialog box in front of figure window	3-45
Functionality being removed or changed	3-45
<b>External Language Interfaces</b>	<b>3-44</b>
C++ Interface: Support for C++ language features	3-46
C++ Interface: Publisher options and analysis	3-46
Java Packages to be removed	3-46
Java Engine: MATLAB value object support	3-46
Python Interface and Engine: Version 3.6 support discontinued	3-46
Perl 5.32.0: MATLAB support on Windows	3-47
<b>Hardware Support</b>	<b>3-48</b>
Support added for IMU sensors	3-48
New functionalities added to Raspberry Pi Resource Monitor app	3-48

<b>Environment</b> .....	<b>4-2</b>
MATLAB Online Accessibility: Use a screen reader to interact with the Command Window and create scripts and functions .....	<b>4-2</b>
Live Editor Images: Add alternative text to images .....	<b>4-2</b>
Live Editor Images: Change the size of images .....	<b>4-2</b>
Live Editor Hyperlinks: Navigate to existing files from a live script or live function using links .....	<b>4-2</b>
Live Editor Export: Export all live scripts and live functions in a folder to a standard format .....	<b>4-2</b>
matlabRelease Object: Query MATLAB Release Information .....	<b>4-3</b>
Query Parallel Functionality: Determine if support for Parallel Computing Toolbox functionality is available .....	<b>4-3</b>
Comparison Tool: Compare text files in MATLAB Online .....	<b>4-3</b>
 <b>Language and Programming</b> .....	 <b>4-4</b>
pattern Object and Functions: Match patterns in text functions .....	<b>4-4</b>
extract Function: Extract substrings from strings .....	<b>4-4</b>
Functions: New validation functions for arguments and properties .....	<b>4-4</b>
underlyingType, isUnderlyingType, and mustBeUnderlyingType Functions: Query the underlying data type of classes .....	<b>4-5</b>
height and width Functions: Return number of rows or columns in an array .....	<b>4-5</b>
Class conversions: Assignment operations convert more classes into built-in data types .....	<b>4-5</b>
Functionality being removed or changed .....	<b>4-6</b>
 <b>Data Analysis</b> .....	 <b>4-7</b>
Implicit Expansion: For calendarDuration, categorical, datetime, and duration arrays, automatically expand dimensions of length 1 when applying element-wise operations and functions .....	<b>4-7</b>
normalize Function: Scale data by interquartile range .....	<b>4-8</b>
groupsummary Function: Summarize data using functions that require multiple input arguments .....	<b>4-8</b>
fillmissing Function and Clean Missing Data Live Editor Task: Specify maximum gap size to fill .....	<b>4-8</b>
Clean Outlier Data Live Editor Task: Define outliers based on percentile thresholds .....	<b>4-8</b>
Functionality being removed or changed .....	<b>4-8</b>
 <b>Data Import and Export</b> .....	 <b>4-10</b>
readstruct and writestruct functions: Read and write structured data in XML files .....	<b>4-10</b>
readlines function: Read the lines in a text file as a string array .....	<b>4-10</b>
Spreadsheet files: Customize formatting when writing data to spreadsheet files with PreserveFormat and AutoFitWidth .....	<b>4-10</b>
imread function and Tiff object: Read images from Aperio SVS and TIFF files containing JPEG2000 compression .....	<b>4-10</b>
ArrayDatastore object: Create datastores from in-memory data .....	<b>4-10</b>

Datastore: Transform multiple datastores using the transform function . . . . .	4-10
FileDatastore object: Shuffle and create subsets of a FileDatastore . . . . .	4-11
writeall function: Write data from text and spreadsheet files to different row groups in Parquet files . . . . .	4-11
fileparts function: Parse file names specified as cell arrays of character vectors and string arrays . . . . .	4-11
Audio devices: Refresh the available audio devices using the audiodevreset function . . . . .	4-11
Audio files and web-based data: Read and write remotely stored audio files using audioread, audiowrite, and audioinfo . . . . .	4-11
HDF5 files and web-based data: Read and write remotely-stored HDF5 files using existing HDF5 functions . . . . .	4-12
HDF5 files: Read and write file names encoded using Unicode characters . . . . .	4-12
Scientific File Format Libraries: NetCDF library upgraded to 4.7.3 . . . . .	4-12
Image File Format Libraries: LibTIFF library upgraded to version 4.1.0 . . . . .	4-12
Bluetooth Interface: Support for communicating with Bluetooth devices . . . . .	4-12
TCP/IP Client Interface: New functions and properties . . . . .	4-13
Serial Port Interface: Improved performance . . . . .	4-13
Functionality being removed or changed . . . . .	4-14
<b>Mathematics . . . . .</b>	<b>4-16</b>
Optimize Live Editor Task: Solve optimization problems interactively . . . . .	4-16
pagetimes Function: Perform matrix multiplication on pages of N-D arrays . . . . .	4-16
pagetranspose and pagectranspose Functions: Transpose pages of N-D arrays . . . . .	4-16
svdsketch Function: Compute SVD factors of low-rank matrix sketch . . . . .	4-16
Functionality being removed or changed . . . . .	4-16
<b>Graphics . . . . .</b>	<b>4-17</b>
bubblechart, bubblechart3, and polarbubblechart Functions: Create bubble charts in 2-D, 3-D, and in polar coordinates . . . . .	4-17
Swarm charts and Scatter objects: Visualize distributions of discrete data . . . . .	4-17
scatter Function: Vary the transparency across all points . . . . .	4-18
tiledlayout and nexttile Functions: Improved placement of legends, and colorbars, and shared decorations . . . . .	4-18
axis Function: Pad axis limits to show plotted data near the limits more clearly . . . . .	4-19
Titles, Subtitles, and Axis Labels: Add subtitles to plots, and align titles and axis labels with the plot box . . . . .	4-20
Data Tips: Customize data tip content on standalone visualizations . . . . .	4-21
turbo Colormap: jet colormap alternative with more perceptually uniform transitions . . . . .	4-22
Colormap Editor: Customize colormaps using modernized interface . . . . .	4-22
boxchart Function: Use color to differentiate between box charts . . . . .	4-23
im2gray and cmap2gray: Convert images and colormaps to grayscale . . . . .	4-23
validatecolor Function: Calculate normalized RGB triplets for color names, hexadecimal color codes, or integer values . . . . .	4-23
Markers: Specify horizontal or vertical line markers for plots . . . . .	4-24
surf and meshc Functions: Specify Z-level for contours on surface and mesh plots . . . . .	4-24



animatedline Function: Create animated lines in polar plots . . . . .	4-25
colororder Function: Control colors in geographic bubble charts . . . . .	4-25
Functionality being removed or changed . . . . .	4-25
<b>App Building . . . . .</b>	<b>4-29</b>
uitable Function: Configure column widths to use weighted variable or to automatically adjust to fit data . . . . .	4-29
scroll Function: Scroll to the top or bottom of a text area programmatically . . . . .	4-29
WindowStyle Property: Create modal UI figures . . . . .	4-29
Icon Property: Specify custom icons for UI figure windows and toolbar push and toggle tools . . . . .	4-29
WordWrap Property: Wrap long text to fit the width of certain UI components . . . . .	4-29
Enable Property: Turn interaction off and on for buttons and panel groups . . . . .	4-30
BackgroundColor Property: Set the background color for grid layouts . . . . .	4-30
Custom Components: Develop your own class of UI components . . . . .	4-30
App Designer: Allow only one running instance of your app at a time . . . . .	4-30
App Designer: Change the stacking order of UI components . . . . .	4-31
App Designer: Add and configure toolbar components on the App Designer canvas . . . . .	4-31
App Designer: Draw UI components on the App Designer canvas . . . . .	4-31
App Designer: Find differences and merge apps . . . . .	4-31
Graphics Support: Create more plots in apps with full support for any type of axes . . . . .	4-32
Graphics Support: Identify coordinates and display text by clicking or tapping . . . . .	4-32
App Capture: Capture user interfaces using exportapp and getframe . . . . .	4-32
App Testing Framework: Perform choose gestures on context menu items . . . . .	4-33
App Testing Framework: Perform drag gestures on axes and UI axes . . . . .	4-33
App Testing Framework: Perform gestures on push tools and toggle tools . . . . .	4-33
Functionality Being Removed or Changed . . . . .	4-33
<b>Performance . . . . .</b>	<b>4-37</b>
sum Function: Improved performance summing the first dimension of numeric arrays . . . . .	4-37
polyfit Function: Improved performance fitting data . . . . .	4-37
accumarray Function: Improved performance with fill values and certain function handles . . . . .	4-38
spdiags Function: Improved performance constructing sparse banded matrices . . . . .	4-39
uilistbox: Improved performance when setting multiple items in a list box . . . . .	4-40
uitree: Improved performance when creating many nodes in a tree . . . . .	4-40
Data Tip Markers: Improved rendering performance of data tip markers in surface plots of large data sets created in UI figures and MATLAB Online . . . . .	4-41
<b>Software Development Tools . . . . .</b>	<b>4-42</b>

Code Compatibility Report: Unsupported Functionality Will Now Issue Warning . . . . .	4-42
Dependency Analyzer: Export to archive and generate a dependency report . . . . .	4-42
Source Control: Improved workflow to set up Git source control . . . . .	4-42
Projects: Change project definition file type and preserve source control history . . . . .	4-42
Unit Testing Framework: Run tests in parallel on clusters and clouds . . . . .	4-42
Unit Testing Framework: Run tests in parallel with standalone applications . . . . .	4-43
Unit Testing Framework: Report the validity of shared test fixtures . . . . .	4-43
App Testing Framework: Perform choose gestures on context menu items . . . . .	4-43
App Testing Framework: Perform drag gestures on axes and UI axes . . . . .	4-43
App Testing Framework: Perform gestures on push tools and toggle tools . . . . .	4-43
Functionality being removed or changed . . . . .	4-44
<b>External Language Interfaces . . . . .</b>	<b>4-46</b>
C++ Interface: Support for nullptr . . . . .	4-46
C++ Interface: Create interface with C++ source files . . . . .	4-46
Python: Version 3.8 support . . . . .	4-46
Python: Terminate Python interpreter and start new one in same MATLAB session . . . . .	4-46
mxCreateString C Matrix API functions: UTF-8 support . . . . .	4-46
MATLAB Data API: Create matlab::data::Object arrays . . . . .	4-46
Compiler support changed for building C++ interfaces, MEX files, and standalone MATLAB engine and MAT-file applications . . . . .	4-47
Functionality being removed or changed . . . . .	4-47
<b>Hardware Support . . . . .</b>	<b>4-48</b>
Live Editor Task: Interactively capture images from USB Webcam interactively and generate MATLAB code in a live script. . . . .	4-48
Functionality being removed or changed . . . . .	4-48

## R2020a

<b>Environment . . . . .</b>	<b>5-2</b>
Profiler Flame Graphs: Investigate and improve the performance of your code visually . . . . .	5-2
Live Editor Loop Execution: Improved performance when running loops in live scripts . . . . .	5-2
Live Editor Animation Output: Improved performance when animating plots in live scripts . . . . .	5-3
Live Editor Responsiveness: Improved performance with extended use . . . . .	5-3
Live Editor Control Value Changes: Run all necessary code on value changes . . . . .	5-3
File Encoding: Save MATLAB code files (.m) and other plain text files as UTF-8 encoded files by default . . . . .	5-3

Multiple Sources in Help Browser: Search MathWorks documentation and custom documentation together in a single browser . . . . .	5-4
Web Documentation: View MathWorks documentation on the web without logging in . . . . .	5-4
Internationalization: UTF-8 as system encoding on Mac and Windows platforms . . . . .	5-5
<b>Language and Programming . . . . .</b>	<b>5-6</b>
switch Function: Compare objects more flexibly . . . . .	5-6
copyfile and movefile Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage . . . . .	5-6
dbup and dbdown Commands: Switch between workspaces with one step . . . . .	5-6
bin2dec and hex2dec Functions: Convert text that includes binary or hexadecimal prefixes and suffixes . . . . .	5-6
dec2bin and dec2hex Functions: Convert negative numbers . . . . .	5-7
complex Function: Create sparse complex arrays . . . . .	5-7
Enumeration classes: Hide member names for compatible name changes . . . . .	5-7
matlab.mixin.SetGet: Set priority for partial property name matching . . . . .	5-7
Class logical conversions: Support logical conversion more flexibly when writing classes . . . . .	5-7
Functionality being removed or changed . . . . .	5-8
<b>Data Analysis . . . . .</b>	<b>5-12</b>
Live Editor Tasks: Interactively manipulate tables and timetables, and generate code . . . . .	5-12
Basic Fitting Tool: Fit lines to plotted data using modernized interface . . . . .	5-12
detrend Function: Ignore NaN values . . . . .	5-13
accumarray Function: Maintain consistent output order on all platforms . . . . .	5-13
leapseconds Function: List all leap seconds used by the datetime data type . . . . .	5-13
timezones Function: Determine IANA Time Zone Database version . . . . .	5-14
renamevars Function: Rename variables in table or timetable . . . . .	5-14
rows2vars and unstack Function: Use naming rule to allow table and timetable variable names with any characters . . . . .	5-14
containsrange, overlapsrange, and withinrange Functions: Determine if timetable row times intersect specified time range . . . . .	5-14
tall Arrays: Operate on tall arrays with more functions, including groupfilter and matches . . . . .	5-15
Functionality Being Removed or Changed . . . . .	5-16
<b>Data Import and Export . . . . .</b>	<b>5-17</b>
Datastores: Write data from datastore to files using writeall . . . . .	5-17
Datastores: Return timetables from tabularTextDatastore and spreadsheetDatastore objects . . . . .	5-17
Datastores: Partition and shuffle TransformedDatastore and CombinedDatastore objects . . . . .	5-17
Datastores: Process files and blocks within files iteratively using FileSet and BlockedFileSet objects . . . . .	5-17
Parquet Files: Control encoding scheme and Parquet version when writing files . . . . .	5-17

Text and Spreadsheet Files: Append, overwrite, or replace data using 'WriteMode' parameter . . . . .	5-18
readtable Function: Uses results of detectImportOptions function by default . . . . .	5-18
textscan, readtable, detectImportOptions, and setvaropts Functions: Read and import hexadecimal and binary literals . . . . .	5-19
h5read and h5readatt: Read non-scalar string data as MATLAB string arrays . . . . .	5-19
h5create and h5write: Write string data to HDF5 files . . . . .	5-19
CDF Library: Upgraded to v3.7.0 . . . . .	5-20
Tiff Object: Read and write the values of the Rational Polynomial Coefficients tag . . . . .	5-20
jsonencode: Customize encoding in MATLAB classes . . . . .	5-20
jsonencode: Encode enumerations . . . . .	5-20
Functionality being removed or changed . . . . .	5-20
<b>Mathematics</b> . . . . .	<b>5-22</b>
nufft and nufftn Functions: Compute nonuniform fast Fourier transforms . . . . .	5-22
sparse Function: Support for integer subscripts and logical aggregation . . . . .	5-22
<b>Graphics</b> . . . . .	<b>5-23</b>
boxchart Function: Visualize grouped numeric data by using box charts . . . . .	5-23
exportgraphics and copygraphics Functions: Save and copy graphics with improved support for publishing workflows . . . . .	5-23
ChartContainer Class: Develop charts that display a tiling of Cartesian, polar, or geographic plots . . . . .	5-24
Tiled Chart Layout: Position, nest, and change the grid size of layouts . . . . .	5-24
pie Function: Specify a numeric format for the percentage labels . . . . .	5-24
Axes Convenience Functions: Pass an array of axes or chart objects to convenience functions such as grid, hold, and box . . . . .	5-25
SeriesIndex and NextSeriesIndex Properties: Control how plots cycle through colors and line styles . . . . .	5-25
colororder Function: Control colors in scatter histograms and parallel plots . . . . .	5-25
pareto Function: Specify the fraction of the cumulative histogram to include . . . . .	5-25
Axes: Control margins for titles and labels by setting the InnerPosition and PositionConstraint properties . . . . .	5-26
Built-In Axes Interactions: Explore data with cursors that show available interactions . . . . .	5-26
Built-In Axes Interactions: Customize built-in interactions on geographic axes . . . . .	5-27
linkdata Function: Open dialog box to specify data sources using new syntax . . . . .	5-27
Functionality being removed or changed . . . . .	5-27
<b>App Building</b> . . . . .	<b>5-30</b>
uicontextmenu Function: Add and configure context menu components in apps and on the App Designer canvas . . . . .	5-30
uitoolbar Function: Add custom toolbars to apps programmatically . . . . .	5-30

Icon Property: Display SVG, animated GIF, or truecolor image array icons in buttons and tree nodes . . . . .	5-30
Mouse Pointer: Change the mouse pointer symbol in apps . . . . .	5-30
Graphics Support: Create annotations, brush data, configure data tips, save and copy graphics . . . . .	5-30
GUIDE to App Designer Migration Tool for MATLAB: Migrate GUIDE apps to App Designer in less time and with fewer manual code updates . . .	5-31
App Testing Framework: Perform press gestures with different selection types . . . . .	5-31
Functionality Being Removed or Changed . . . . .	5-31
<b>Performance . . . . .</b>	<b>5-33</b>
Live Editor Loop Execution: Improved performance when running loops in live scripts . . . . .	5-33
Live Editor Animation Output: Improved performance when animating plots in live scripts . . . . .	5-33
datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting . . . . .	5-34
datetime Data Type Format Parsing: Improved performance when parsing format of text inputs . . . . .	5-35
table Data Type Indexing: Improved performance when assigning elements by subscripting into table variables . . . . .	5-35
Subscripted Reference: Improved performance for struct arrays stored in a property of an object . . . . .	5-36
imread Function: Improved performance in reading JPEG images . . . . .	5-37
readmatrix Function: Improved performance in reading data . . . . .	5-37
ode15s, ode23t, and ode15i Solvers: Improved performance solving differential equations . . . . .	5-38
transpose and ctranspose Functions: Improved performance on large arrays . . . . .	5-39
ordschur and ordqz Functions: Improved performance operating on large matrices . . . . .	5-39
sparse Function: Improved performance constructing sparse matrices . .	5-40
interp1 Function: Faster interpolation for small problem sizes . . . . .	5-40
assert Function: Improved performance for most common use cases . . . .	5-41
nexttile Function: Improved performance when creating several axes in a tiled chart layout . . . . .	5-41
App Designer Code View: Improved performance when displaying and editing code in App Designer . . . . .	5-42
Graphics Rendering in UI Figures: Improved graphics rendering performance on large data sets in UI figures . . . . .	5-43
Data Tip Markers: Improved rendering performance of data tip markers in line plots of large data sets created in UI figures and MATLAB Online . . . . .	5-45
Icon Property: Improved rendering performance for buttons and tree nodes with icons . . . . .	5-45
Functionality being removed or changed . . . . .	5-46
<b>Software Development Tools . . . . .</b>	<b>5-47</b>
Dependency Analyzer: Improved navigation, filtering, and highlighting for project dependencies . . . . .	5-47
Project Checks: Run all project checks programmatically . . . . .	5-48
Project API: Get latest Git revision programmatically . . . . .	5-48
Unit Testing Framework: Add custom details to TestResult objects . . . . .	5-49

Unit Testing Framework: Assert that test session ran with no failure . . .	5-49
Unit Testing Framework: Run tests from the Live Editor toolstrip . . . . .	5-49
Unit Testing Framework: Generate test reports including test tags . . . . .	5-49
App Testing Framework: Perform press gestures with different selection types . . . . .	5-49
Mocking Framework: Add events to mock objects . . . . .	5-49
Mocking Framework: Specify when framework should do nothing . . . . .	5-50
Functionality being removed or changed . . . . .	5-50
<b>External Language Interfaces . . . . .</b>	<b>5-51</b>
C++ Interface: MATLAB data type for C++ array and std::vector . . . . .	5-51
C++ Interface: Supported data types . . . . .	5-51
C++ Interface: Lifetime management of C++ objects . . . . .	5-52
MATLAB Data Array: Support for N-D row-major memory layout . . . . .	5-52
MATLAB COM Server: Register MATLAB without administrative privileges . . . . .	5-52
Java interface: MATLAB support for OpenJDK™ 8 (Hot Spot) . . . . .	5-52
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications . . . . .	5-52
Functionality being removed or changed . . . . .	5-52
<b>Hardware Support . . . . .</b>	<b>5-54</b>
MATLAB Support Package for Ryze Tello Drones: Control Ryze Tello drone from MATLAB and acquire sensor and image data . . . . .	5-54
Support added for Raspberry Pi 4B model board . . . . .	5-54
Deploy deep learning applications on Raspberry Pi hardware . . . . .	5-54
Read GPS Data from GPS Receiver Connected to Arduino Hardware . . . . .	5-54
Use BNO055 Sensor with Sensor Fusion and Tracking Toolbox, and Navigation Toolbox to Estimate Orientation . . . . .	5-55
Enable Code Generation of MATLAB Arduino Functions Inside a MATLAB Function Block for I2C and SPI . . . . .	5-55
Functionality being changed or removed . . . . .	5-55

## R2019b

<b>Environment . . . . .</b>	<b>6-2</b>
Live Editor Tasks: Add interactive tasks to live scripts to explore parameters and automatically generate code . . . . .	6-2
Live Editor Output: Animate plots to show changes in data over time . . . . .	6-4
Live Editor Output: Adjust the width of columns in tables . . . . .	6-4
Live Editor Output: Scroll through and copy data in arrays such as cell arrays, object arrays, and struct arrays . . . . .	6-4
Live Editor Export: Customize figure format as well as document paper size, orientation, and margins when exporting . . . . .	6-5
Live Editor Code: Duplicate one or more lines of code . . . . .	6-5
Live Editor Code: Suppress Code Analyzer warning messages . . . . .	6-6
Live Editor Debugging: Set breakpoints for anonymous functions . . . . .	6-6
Live Editor Internationalization: Add non-English language such as Chinese, Japanese, and Korean characters on Windows and macOS Platforms . . . . .	6-6

Add-On Manager: Update MATLAB and other installed add-ons . . . . .	6-6
Add-On Manager: Programmatically manage add-ons by name . . . . .	6-6
Settings: Create persistent settings for custom apps, toolboxes, and across MATLAB sessions . . . . .	6-7
MATLAB Drive: Share folders and collaborate with others . . . . .	6-7
Functionality being removed or changed . . . . .	6-7
<b>Language and Programming . . . . .</b>	<b>6-8</b>
size Function: Find lengths of multiple array dimensions at a time . . . . .	6-8
matches Function: Determine if input strings are equal . . . . .	6-8
Hexadecimal and Binary Numbers: Specify numbers using hexadecimal and binary literals . . . . .	6-8
Indexing: Use dot indexing into function calls . . . . .	6-8
System object authoring improvements: Property validation support and simplified class inheritance . . . . .	6-8
Function Input Arguments: Declare function input arguments to restrict values . . . . .	6-9
namedargs2cell Function: Convert structure containing name-value pairs to cell array . . . . .	6-9
delete, dir, isfile, isfolder, and what Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage . . . . .	6-9
Suggested Corrections: Correct errors with two new classes . . . . .	6-9
error Function: Provide suggested fix for uncaught exception . . . . .	6-9
Functionality being removed or changed . . . . .	6-9
<b>Data Analysis . . . . .</b>	<b>6-14</b>
Live Editor Tasks: Interactively preprocess data and generate code . . . . .	6-14
groupfilter Function: Filter data in a table, timetable, or matrix by group . . . . .	6-14
datetime Data Type: Detect formats with fractional seconds when converting text that represents dates and times . . . . .	6-15
table and timetable Data Types: Variable names can have any characters, including spaces and non-ASCII characters . . . . .	6-15
tall Arrays: Operate on tall arrays with more functions, including setdiff, xcorr, and outerjoin . . . . .	6-16
tall Arrays: Avoid running out of memory due to temporary copies of data . . . . .	6-17
<b>Data Import and Export . . . . .</b>	<b>6-18</b>
detectImportOptions Function: Specify the type of import options for delimited or fixed-width text files . . . . .	6-18
table and timetable Data Types: Read and write tabular data that has variable names containing any characters, including spaces and non- ASCII characters . . . . .	6-18
sheetnames Function: Get names of worksheets from spreadsheet file . . . . .	6-18
VideoReader Object: Read frames in videos using frame index or time . . . . .	6-18
VideoReader Object: Improved performance in generated code with row- major layout . . . . .	6-19
Import Tool: Generate simpler code when importing from fixed-width text files . . . . .	6-20
save Function: Save workspace variables to a MAT-file version 7 without compression . . . . .	6-20

xmlread Function: Prevent reading of XML files that contain DOCTYPE declarations	6-20
imread Function: Supports reading specified images from PGM, PBM, or PPM file formats	6-20
Scientific File Format Libraries: CFITSIO Library upgraded to version 3.450	6-20
Scientific File Format Libraries: LibTIFF Library upgraded to version 4.0.10	6-21
RESTful Functions: Support for authentication	6-21
tcpclient, read, and write Functions: Generate C and C++ code	6-21
Bluetooth Low Energy Interface: Support for scanning and interacting with peripheral devices	6-21
Serial Port Devices: New functions and properties	6-21
Functionality being removed or changed	6-22
<b>Mathematics</b>	<b>6-8</b>
makima Function: Perform modified Akima cubic Hermite interpolation	6-26
<b>Graphics</b>	<b>6-14</b>
Chart Container Class: Develop your own class of charts	6-27
tiledlayout and nexttile Functions: Create configurable layouts of plots in a figure	6-27
colororder Function: Control the colors in plots	6-28
Bar Charts: Create bar charts with improvements for stacking and locating the tips of bars	6-28
Data Tips: Create and customize data tips	6-28
dataTipInteraction Function: Pin data tips at cursor location	6-29
Axes Toolbar: Save or copy contents of axes as image	6-29
parallelplot Function: Zoom, pan, and rearrange coordinates interactively	6-29
Property Inspector: Update axis tick values and labels using clipboard data	6-29
Image Interpolation: Select an interpolation method for displaying images	6-30
legend Function: Create unlimited legend entries and specify categorical arrays	6-30
pcolor Function: Specify categorical, datetime, and duration data	6-30
Geographic Plots: Plot data on high-zoom-level basemaps	6-30
Geographic Plots: Create plots with improved basemap appearance	6-31
Geographic Axes: Display animations using comet or animatedline	6-32
Geographic Bubble Charts: Create charts with improved layout	6-32
Functionality being removed or changed	6-32
<b>App Building</b>	<b>6-18</b>
uistyle Function: Create styles for rows, columns, or cells in a table UI component	6-36
uigridlayout Function: Configure grid rows and columns to adjust automatically to fit components	6-36
uitable Function: Sort table UI components interactively when using logical, numeric, string, or cell arrays	6-36
uihtml Function: Embed HTML, JavaScript, or CSS content in apps and on the App Designer canvas	6-36



App Designer: Convert components in a UI figure or container from pixel-based positioning to a grid layout manager . . . . .	6-37
App Designer: Convert an existing app into an auto-reflowing app . . . . .	6-37
App Designer: Suppress Code Analyzer warning messages . . . . .	6-37
App Designer: Open App Designer from the MATLAB toolstrip . . . . .	6-38
App Testing Framework: Perform gestures on polar axes and UI images . . . . .	6-38
Functionality being removed or changed . . . . .	6-38
<b>Performance . . . . .</b>	<b>6-41</b>
table Data Type Indexing: Improved performance when assigning elements by subscripting into large table variables . . . . .	6-41
datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting into large arrays . . . . .	6-42
datetime Data Type Indexing: Improved performance when referring or assigning to date and time components of datetime arrays . . . . .	6-43
uitable Function: Faster performance when data type is numeric, logical, or a cell array of character vectors . . . . .	6-44
unzip and gunzip Functions: Improved performance when extracting contents of zip files and GNU zip files . . . . .	6-44
<b>Software Development Tools . . . . .</b>	<b>6-46</b>
Unit Testing Framework: Run tests in parallel with your custom plugins . . . . .	6-46
Unit Testing Framework: Validate count in string constraints . . . . .	6-46
Performance Testing Framework: Visually compare two TimeResult arrays . . . . .	6-46
App Testing Framework: Perform gestures on polar axes and images . . . . .	6-46
Projects: Delete project definition files . . . . .	6-47
Compare Git Branches: Show differences and save copies . . . . .	6-47
Functionality being removed or changed . . . . .	6-47
<b>External Language Interfaces . . . . .</b>	<b>6-48</b>
C++ Interface: Options for publishing C++ interface library . . . . .	6-48
C++ Interface: nullptr supported as output argument . . . . .	6-48
C++ Interface: Read-only (const) object support . . . . .	6-48
Java Interface: JRE version 1.8.0_202 support . . . . .	6-48
Out-of-Process Execution of C++ MEX Functions: Customize environment variables . . . . .	6-48
HTTP Web Services: Server authentication support for NTLM and Kerberos protocols . . . . .	6-48
HTTP Web Services: Timeout options . . . . .	6-49
Python Interface: Execute Python functions out of process . . . . .	6-49
Python Interface and Engine: Version 3.5 support discontinued . . . . .	6-49
Perl 5.30.1: MATLAB support on Windows . . . . .	6-49
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications . . . . .	6-50
Functionality being removed or changed . . . . .	6-50

<b>Environment</b> .....	<b>7-2</b>
Live Editor Controls: Add check boxes, edit fields, and buttons to set variable values and run the live script .....	7-2
Live Editor Controls: Specify what code to run when a control value changes .....	7-2
Live Editor Controls: Hide code when sharing and exporting live scripts with interactive controls .....	7-2
Live Editor Export: Save live scripts and functions as Microsoft Word documents .....	7-2
Live Editor Output: Enable animations in plots to show changes in data over time .....	7-3
Live Editor Output: Interactively clean categorical data and filter datetime and duration variables in table output .....	7-4
Live Editor Output: Interactively change the data type of variables in table output .....	7-4
Live Editor Functions: Automatically convert selected code to a function .....	7-5
MATLAB Online: Share folders and collaborate with others .....	7-5
Projects: Organize, manage, and share your work using projects .....	7-5
MATLAB Startup: Execute MATLAB script or function non-interactively ..	7-5
Toolbox Packaging: Install required add-ons with custom toolboxes .....	7-6
 <b>Language and Programming</b> .....	 <b>7-7</b>
append Function: Combine strings .....	7-7
MException class: Provide a suggested fix for an uncaught exception ....	7-7
Functionality being removed or changed .....	7-7
 <b>Data Analysis</b> .....	 <b>7-10</b>
xcorr and xcov Functions: Compute cross-correlation and cross-covariance in core MATLAB .....	7-10
detrend Function: Remove piecewise polynomial trends, set continuity requirements, and specify sample points .....	7-10
groupcounts Function: Count the number of group elements for arrays, tables, and timetables .....	7-10
grouptransform Function: Transform array data by group .....	7-10
filloutliers, isoutlier, and rmoutliers Functions: Detect outliers using percentiles .....	7-10
fillmissing and filloutliers Functions: Fill missing and outlier data using modified Akima interpolation .....	7-10
fillmissing Function: Specify missing value locations .....	7-10
min and max Functions: Return index information when operating on more than one dimension and specify linear indices .....	7-11
tall Arrays: Write custom sliding-window algorithms to operate on tall arrays .....	7-11
tall Arrays: Operate on tall arrays with more functions, including groupcounts, intersect, and svd .....	7-11
Functionality Being Removed or Changed .....	7-12
 <b>Data Import and Export</b> .....	 <b>7-13</b>

readmatrix, readvars, and readcell Functions: Read tabular data as a matrix, variables, or a cell array . . . . .	7-13
writematrix and writecell functions: Write tabular data from a matrix or cell array to a text or spreadsheet file . . . . .	7-13
readtimetable and writetimetable Functions: Read and write timetables . . . . .	7-13
detectImportOptions Function: Improve detection of import options for text and spreadsheet files . . . . .	7-13
parquetread, parquetwrite, and parquetinfo Functions: Read, write, and get information from Parquet files . . . . .	7-14
write Function: Write tall arrays to Parquet files . . . . .	7-14
Import Tool: Generate improved code when importing from text files . . .	7-14
thingSpeakRead and thingSpeakWrite Functions: Read or write data to the ThingSpeak IoT platform . . . . .	7-14
writetable and imwrite Functions: Write to web-based storage services like Amazon Web Services and Azure Blob Storage . . . . .	7-14
ParquetDatastore Object: Create a datastore for a collection of Parquet files . . . . .	7-15
ImageDatastore Object: Create a subset of an existing datastore . . . . .	7-15
DsFileSet Object: Create a subset of a file collection . . . . .	7-15
FileDatastore Object: Read large files by importing the file in smaller portions . . . . .	7-15
Datastores: Combine and transform datastores . . . . .	7-15
Custom Datastore: Read Hadoop based data from files, databases, and other non-file-based locations . . . . .	7-15
VideoReader function: Generate C and C++ code . . . . .	7-15
ind2rgb function: Generate C and C++ code . . . . .	7-16
Scientific File Format Libraries: NetCDF Library upgraded to version 4.6.1 . . . . .	7-16
web function: Open external sites in system browser instead of MATLAB browser . . . . .	7-16
Functionality being removed or changed . . . . .	7-16
<b>Mathematics</b> . . . . .	<b>7-19</b>
Solve assignment problem with matchpairs and equilibrate . . . . .	7-19
graph and digraph Objects: Construct graphs with categorical nodes . . .	7-19
<b>Graphics</b> . . . . .	<b>7-20</b>
parallelplot Function: Visualize tabular or matrix data with multiple columns by using a parallel coordinates plot . . . . .	7-20
Data Tips: Pin and customize data tips in charts . . . . .	7-20
Axes Interactions: Customize chart interactions such as dragging to pan or scrolling to zoom . . . . .	7-20
Ruler Panning: Pan an axis to change its limits without having to use the pan tool . . . . .	7-21
Property Inspector: Navigate and control visibility of graphics objects interactively . . . . .	7-21
Geographic Plots: Geographic rulers, scale bar, CurrentPoint, and ginput . . . . .	7-21
Graphics Export: Export axes with tighter cropping using the axes toolbar . . . . .	7-22
Chart Resizing: Resize charts with improved layouts . . . . .	7-22
Colors Values: Specify colors using hexadecimal color codes . . . . .	7-23

Categorical Values: Specify categorical arrays for functions and objects that use lists of text . . . . .	7-23
rendererinfo Function: Get renderer information for any axes . . . . .	7-23
Functionality being removed or changed . . . . .	7-23
<b>App Building . . . . .</b>	<b>7-25</b>
uiimage Function: Display an icon, logo, or picture in apps and on the App Designer canvas . . . . .	7-25
uitable Function: Sort tables interactively when using table arrays . . . . .	7-25
Auto Resize: Automatically resize components when an app is made smaller . . . . .	7-25
Scrolling Grids: Create apps with scrollable grids . . . . .	7-25
App Designer: Create apps that automatically reflow content based on device size . . . . .	7-25
App Designer: Add and configure a grid layout manager on the App Designer canvas . . . . .	7-25
App Designer: Rearrange the order of callbacks . . . . .	7-26
App Designer: Create new apps using App Designer Start Page options . . . . .	7-26
App Designer: Control font, code, and autosave settings using MATLAB Preferences . . . . .	7-26
App Designer: Access context-sensitive help in Code View . . . . .	7-26
App Designer: Zoom in App Designer . . . . .	7-26
Graphics Support: Explore data using axes toolbar and data tips in apps created with the uifigure function . . . . .	7-27
Deployed Web Apps: Share resizable apps or create apps that open web pages . . . . .	7-27
MATLAB Online: Create and edit App Designer apps using MATLAB Online . . . . .	7-27
App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures . . . . .	7-27
App Testing Framework: Perform press gesture on axes, UI axes, and UI figures . . . . .	7-28
App Testing Framework: Perform type gesture on date picker objects . . . . .	7-28
Functionality Being Removed or Changed . . . . .	7-28
<b>Performance . . . . .</b>	<b>7-29</b>
MATLAB and Simulink startup on macOS platforms . . . . .	7-29
sortrows Function: Sort rows of large matrices faster . . . . .	7-29
uitable Function: Faster performance using table arrays . . . . .	7-29
<b>Software Development Tools . . . . .</b>	<b>7-30</b>
checkcode Function: Get the modified cyclomatic complexity of functions . . . . .	7-30
Source Control Integration: Synchronise MATLAB Git status with external Git clients . . . . .	7-30
Unit Testing Framework: Display code coverage metrics in HTML format . . . . .	7-30
Unit Testing Framework: Specify sources for collections of code coverage data with runtests . . . . .	7-30
Unit Testing Framework: runperf collects more samples to achieve its target margin of error . . . . .	7-30
Unit Testing Framework: Return performance test results as TimeResult arrays . . . . .	7-30

Unit Testing Framework: Load previously saved MeasurementResult objects as DefaultMeasurementResult . . . . .	7-31
Unit Testing Framework: Use matlab.unittest.fixtures.Fixture.onFailure method only in subclasses . . . . .	7-31
Unit Testing Framework: Compare tables that contain no rows . . . . .	7-31
Unit Testing Framework: Create test suite array from tests in project . . .	7-32
Unit Testing Framework: Run tests from files in project using runtests or testsuite . . . . .	7-32
Unit Testing Framework: Specify verbosity enumeration as a string or character vector . . . . .	7-32
App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures . . . . .	7-32
App Testing Framework: Perform press gesture on axes, UI axes, and UI figures . . . . .	7-32
App Testing Framework: Perform type gesture on date picker objects . . .	7-32
Mocking Framework: Create mocks for classes that use custom metaclasses . . . . .	7-32
Mocking Framework: Create mocks for classes that use property validation . . . . .	7-33
Mocking Framework: Specify which methods to mock . . . . .	7-33
Functionality being removed or changed . . . . .	7-33
<b>External Language Interfaces . . . . .</b>	<b>7-34</b>
C++: Use C++ classes from third-party libraries in MATLAB . . . . .	7-34
Python: Version 3.7 support . . . . .	7-34
Python engine: Data type support . . . . .	7-34
C++ MEX: Execute MEX function out of process . . . . .	7-34
MEX functions: Use customer version of Boost library . . . . .	7-34
MATLAB Data Array: Support for row-major memory layout . . . . .	7-34
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications . . . . .	7-35
<b>Hardware Support . . . . .</b>	<b>7-36</b>
MATLAB Support Package for Parrot Drones: Control Parrot Mambo FPV drone from MATLAB and acquire sensor data . . . . .	7-36
Deploy Sense HAT functions on Raspberry Pi hardware . . . . .	7-36
Functionality being changed or removed . . . . .	7-36

## R2018b

<b>Desktop . . . . .</b>	<b>8-2</b>
Live Editor: Organize live scripts using additional subheading styles . . . .	8-2
Live Editor: Navigate within a live script using internal hyperlinks . . . .	8-2
Live Editor: Filter table output interactively, and then add the generated code to the live script . . . . .	8-2
Live Editor: Create new and open existing live scripts faster . . . . .	8-2
Live Editor: Change case of text or code . . . . .	8-2
Comparison Tool: Merge two versions of a live script or function . . . . .	8-3

Add-On Manager: Install and manage multiple versions of a custom toolbox	8-3
Add-On Manager: Save add-ons to new default location	8-3
Documentation: View MATLAB documentation in Spanish	8-4
<b>Language and Programming</b>	<b>8-6</b>
string Arrays: Use string arrays in MATLAB, Simulink, and Stateflow	8-5
convertContainedStringsToChars Function: Convert string arrays at any level of cell array or structure	8-5
Enumerations: Improved performance of set operations with enumerations	8-5
WSDL Web Services Documents: Required Tools Update	8-5
Functionality being removed or changed	8-5
<b>Mathematics</b>	<b>8-19</b>
boundaryshape Function: Create a polyshape object from a 2-D triangulation	8-7
polyshape Objects: Specify when to keep collinear points when creating a polyshape	8-7
RandStream Objects: Generate random numbers using Threefry and Philox algorithms	8-7
GraphPlot Object: Customize node and edge labels with font properties	8-7
sinpi and cospi Functions: Compute the sine and cosine of multiples of $\pi$	8-8
<b>Graphics</b>	<b>8-21</b>
Axes Interactions: Explore data with panning, zooming, data tips, and 3-D rotation enabled by default	8-9
Axes Toolbar: Access and customize a data exploration toolbar for each Axes object	8-9
Geographic Plots: Create line, scatter, and point density plots on interactive maps and control properties of a geographic axes	8-9
stackedplot Function: Plot variables of a table or timetable for comparison using a common x-axis	8-10
scatterhistogram Function: Visualize grouped data as a scatter plot with marginal histograms	8-10
sgtitle Function: Create a title for a grid of subplots	8-10
xline and yline Functions: Add vertical or horizontal lines to a plot	8-10
imtile Function: Combine multiple image frames into one rectangular tiled image	8-11
Data Tips: Use TeX or LaTeX markup in data tips with improved visual appearance	8-11
Functionality being removed or changed	8-12
<b>Data Import and Export</b>	<b>8-14</b>
Import Tool: Generate improved code when importing from spreadsheets	8-13
Web-Based Data: Read from web-based data sources like Amazon Web Services and Azure Blob Storage using readtable, detectImportOptions, spreadsheetDatastore, imread, and imfinfo	8-13

write Function: Write tall arrays in a variety of formats to local or remote locations . . . . .	8-13
stlread and stlwrite Functions: Read from and write to STL (Stereolithography) files for triangulations . . . . .	8-14
TabularTextDatastore Object: Import data containing dates and times from non-English locales . . . . .	8-14
readtable and writetable Functions: Read or write spreadsheet files without initiating Microsoft Excel for Windows on Windows platforms . . . . .	8-14
readtable Function: Manage the import of empty fields using import options . . . . .	8-14
Scientific File Format Libraries: CFITSIO Library upgraded to version 3.420 . . . . .	8-14
Functionality being removed or changed . . . . .	8-14
<b>Data Analysis . . . . .</b>	<b>8-11</b>
Vector Dimension Argument: Operate on multiple dimensions at a time for selected reduction functions . . . . .	8-17
grouptransform Function: Transform table or timetable data by groups . . . . .	8-17
groupsummary Function: Perform group summary computations on matrices . . . . .	8-17
tall Arrays: Write custom algorithms to operate on tall arrays . . . . .	8-17
tall Arrays: Operate on tall arrays with more functions, including conv2, wordcloud, and groupsummary . . . . .	8-18
rmoutliers Function: Remove outliers in an array, table, or timetable . . . . .	8-18
islocalmin and islocalmax Functions: Specify a range of data for prominence computation . . . . .	8-18
Table and Timetable Metadata: Store custom metadata for each variable . . . . .	8-19
timetable Data Type: Save memory when storing row times with regular time steps . . . . .	8-19
timerange Function: Specify unit of time to define time range . . . . .	8-19
convertvars Function: Convert table or timetable variables to specified data type . . . . .	8-19
table, timetable, and addvars Functions: Use single quotes for input names, not double-quoted strings . . . . .	8-19
Functionality Being Removed or Changed . . . . .	8-19
<b>App Building . . . . .</b>	<b>8-31</b>
App Designer: Add and configure date selection components on the App Designer canvas . . . . .	8-21
App Designer: Unified property inspector in Design View and Code View . . . . .	8-21
App Designer: Expand and collapse sections of code in Code View . . . . .	8-21
App Designer: Export apps as code files . . . . .	8-21
App Designer: Locate errors and warnings in your code with the Code Analyzer message bar . . . . .	8-21
App Designer: Program apps faster using improved code suggestions and completions . . . . .	8-21
App Designer: Control App Designer Code View settings using MATLAB preferences . . . . .	8-21
uigridlayout Function: Configure app layouts using a grid layout manager . . . . .	8-22
Scrolling Containers: Enable scrolling for figure, panel, tab, and button group containers . . . . .	8-22

Figure Interactions: Create apps with custom mouse and keyboard interactions using figures created with the uifigure function . . . . .	8-22
Graphics Support: Integrate plots into an app using the axes, polaraxes, and geoaxes functions . . . . .	8-22
Tooltips: Create custom tooltips for UI components in apps . . . . .	8-22
Deployed Web Apps: Access files in deployed web apps using the uigetfile and uiputfile functions . . . . .	8-23
Running Apps in Browsers: Use most modern browsers to run apps in MATLAB Online or as deployed web apps . . . . .	8-23
uisetcolor Function: Select custom colors interactively . . . . .	8-23
Functionality Being Removed or Changed . . . . .	8-23
<b>Performance . . . . .</b>	<b>8-36</b>
Startup: Increased speed of MATLAB startup . . . . .	8-24
Execution Engine: Index into large arrays with improved performance when using the colon operator . . . . .	8-24
Execution Engine: Faster calls to built-in functions . . . . .	8-24
Live Editor: Create new and open existing live scripts faster . . . . .	8-24
Enumerations: Improved set function performance with enumerations . . . . .	8-24
Building Apps: Faster canvas interactions in App Designer . . . . .	8-24
Running Apps: Faster startup time for apps . . . . .	8-24
sort Function: Sort matrices and arrays faster . . . . .	8-24
<b>Hardware Support . . . . .</b>	<b>8-48</b>
MATLAB Online: Communicate with Raspberry Pi hardware board from MATLAB Online . . . . .	8-25
Deploy a MATLAB function on Raspberry Pi hardware . . . . .	8-25
iOS and Android Sensors: Acquire sensor data when your device does not have network access . . . . .	8-25
iOS and Android Sensors: Upload sensor logs from the device to MATLAB Drive . . . . .	8-26
<b>Advanced Software Development . . . . .</b>	<b>8-44</b>
Tab Completion: Validate function signature file with validateFunctionSignaturesJSON function . . . . .	8-27
Tab Completion: JSON parser for functionSignatures.json upgrade . . . . .	8-27
Java SE 8: MATLAB support, providing improved security and access to new Java features . . . . .	8-27
Python Interface: Pass multidimensional numeric or logical arrays between MATLAB and Python . . . . .	8-27
C++ MEX API: Call MATLAB asynchronously from within a MEX file using the C++ API . . . . .	8-27
Unit Testing Framework: Run tests in parallel with more plugins and more intelligent scheduling . . . . .	8-28
Unit Testing Framework: Use external parameters in parameterized test . . . . .	8-28
Unit Testing Framework: Sort test suite based on shared fixtures . . . . .	8-28
Unit Testing Framework: Explicitly control output display detail and logged diagnostic level . . . . .	8-28
Unit Testing Framework: Configure detail level of output diagnostics . . . . .	8-29
Unit Testing Framework: Compare values faster when using constraints . . . . .	8-29
App Testing Framework: Programmatically choose tree node . . . . .	8-30



Performance Testing Framework: Measure execution time of fast code more accurately with the <code>TestCase.keepMeasuring</code> method . . . . .	8-30
Mocking Framework: Invoke function upon mocked method call . . . . .	8-30
Mocking Framework: Verify interactions on mock occurred in order . . . .	8-30
Mocking Framework: Clear history of recorded mock object interactions . . . . .	8-30
<code>matlab.test.behavior</code> : Missing class: Verify class satisfies missing-value behavior contract . . . . .	8-30
MEX Functions: Build Fortran MEX Files with Interleaved Complex API . . . . .	8-31
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications . . . . .	8-31
System objects: Flexible requirements for inputs when calling System objects . . . . .	8-32
System object authoring: Use enumerations to define finite property lists in System objects . . . . .	8-32
Reference Architecture: Deploy and run MATLAB on Amazon Web Services (AWS) and Microsoft Azure . . . . .	8-32
Git Stashes: Store uncommitted changes for later use . . . . .	8-32
Functionality being removed or changed . . . . .	8-32

## R2018a

<b>Desktop</b> . . . . .	9-2
Live Editor: Create live functions with richly formatted documentation, including equations and images . . . . .	9-2
Live Editor: Debug live functions and scripts . . . . .	9-2
Live Editor: Add sliders and drop-down lists to control variable values in a live script . . . . .	9-2
Live Editor: Sort table data interactively . . . . .	9-3
Live Editor: Create a table of contents and add formatted code examples . . . . .	9-3
Live Editor: Select and edit a rectangular area of code . . . . .	9-3
Add-Ons Explorer: Browse by category to discover convenient, helpful add-ons . . . . .	9-4
Comparison Tool: Find differences in live scripts and functions . . . . .	9-4
Favorites: Rerun favorite commands . . . . .	9-4
Toolbox Packaging: Specify portability information for custom toolboxes . . . . .	9-4
<b>Language and Programming</b> . . . . .	9-5
Empty Arrays: Create complex empty arrays using functions such as <code>zeros</code> and <code>ones</code> . . . . .	9-5
Code Compatibility Report: Generate compatibility report from Current Folder browser . . . . .	9-5
timer Object: Access properties with multilevel indexing . . . . .	9-5
Functionality being removed or changed . . . . .	9-6
<b>Mathematics</b> . . . . .	9-15

graph and digraph Objects: Work with multigraphs that have multiple edges between two nodes . . . . .	9-15
graph and digraph Objects: Calculate component sizes and weighted adjacency matrices . . . . .	9-16
GraphPlot Object: Visualize graphs with additional options for 'force', 'force3', and 'circle' layouts . . . . .	9-16
polyshape Objects: Analyze polygons with turningdist, nearestvertex, and overlaps functions . . . . .	9-16
polyshape Objects: Return vertex map and accept arrays with compatible sizes for intersect, subtract, union, and xor functions . . . . .	9-16
polybuffer Function: Create buffer around points or lines . . . . .	9-17
triangulation Objects: Find neighboring vertices and locations of query points with improved performance . . . . .	9-17
ode45 Function: Solve nonstiff differential equations faster . . . . .	9-17
<b>Graphics . . . . .</b>	<b>9-18</b>
Axes Object: View axes at small size with improved layout, limit selection, and font scaling . . . . .	9-18
Axes Object: Map data values to colormap using linear or logarithmic scale . . . . .	9-18
Legend Object: Create legends with multiple columns . . . . .	9-18
heatmap Function: Zoom and pan data, display data tips, and sort rows and columns interactively . . . . .	9-18
geobubble Function: Explore with interactive data tips and a scale bar . . . . .	9-18
Axes Toolbar: Add toolbars to your axes for quick access to pan, zoom, and other data exploration tools . . . . .	9-19
Property Inspector: Modify graphics interactively with an improved property inspector . . . . .	9-19
Polygon Object: Control color and transparency of hole edges . . . . .	9-19
Functionality being removed or changed . . . . .	9-20
<b>Data Import and Export . . . . .</b>	<b>9-21</b>
readtable Function: Specify the number of rows to read from a text file using import options . . . . .	9-21
readtable Function: Easily manage prefixes and suffixes from data using import options . . . . .	9-21
preview Function: Preview first 8 rows of a table in a file without importing the full table . . . . .	9-21
imageDatastore Function: Work with millions of images with improved memory usage and performance . . . . .	9-21
Datastore Functions: Seamlessly work with datasets stored on cloud and local machines . . . . .	9-22
Datastore Functions: Read HDFS data more easily when using Hortonworks or Cloudera . . . . .	9-22
readtable, detectImportOptions, datastore, and tabularTextDatastore Functions: Automatically detect and return duration data in text files . . . . .	9-22
detectImportOptions Function: Control import properties of duration data . . . . .	9-22
VideoReader Function: Read video files faster on all platforms . . . . .	9-23
VideoWriter Function: Write video files faster on all platforms . . . . .	9-23
openDiskFile Function: Read data files in FITS (Flexible Image Transport System) data format . . . . .	9-23
webwrite Function: Support for NTLM authentication . . . . .	9-23

Functionality being removed or changed . . . . .	9-23
<b>Data Analysis</b> . . . . .	<b>9-25</b>
groupsummary Function: Group and discretize data for summary operations on table and timetable variables . . . . .	9-25
Table and Timetable Variables: Add, delete, and rearrange column-oriented variables with the functions addvars, removevars, movevars, splitvars, mergevars, rows2vars, and inner2outer . . . . .	9-25
Preallocated Tables and Timetables: Initialize table and timetable variables so that they have specified sizes and data types . . . . .	9-25
Regular Timetables: Create regularly spaced timetables using a time step or sampling rate . . . . .	9-25
retime and synchronize Functions: Synchronize timetables to a time step or sampling rate that you specify . . . . .	9-26
duration Arrays: Create duration arrays from text that represents elapsed times . . . . .	9-26
normalize Function: Normalize array, table, and timetable data . . . . .	9-26
tall Arrays: Operate on tall arrays with more functions, including smoothdata, find, and isoutlier . . . . .	9-26
tall Array Indexing: Use tall numeric arrays to index the first dimension . . . . .	9-26
tall Arrays: Solve linear systems $Ax = b$ . . . . .	9-26
tall Arrays: Return group labels with findgroups . . . . .	9-27
tall Arrays: Set date and time components of tall datetime and tall duration arrays . . . . .	9-27
tall Arrays: Set properties of tall tables and tall timetables . . . . .	9-27
Functionality being removed or changed . . . . .	9-27
<b>App Building</b> . . . . .	<b>9-29</b>
App Designer: Create deployed web apps using MATLAB Compiler . . . . .	9-29
App Designer: Add and configure tree components on the App Designer canvas . . . . .	9-29
App Designer: Select from recently used argument sets when running apps with input arguments . . . . .	9-29
App Designer: Edit axes title and label directly in the canvas . . . . .	9-29
GUIDE: Migrate GUIDE apps to App Designer . . . . .	9-29
App Testing Framework: Author automated tests for App Designer apps . . . . .	9-29
Figure Objects: Maximize and minimize figures programmatically . . . . .	9-29
uitable Function: Specify data as table array . . . . .	9-30
uidatepicker Function: Add date selection controls to apps . . . . .	9-30
uiprogressdlg Function: Create modal in-app progress dialog boxes to apps . . . . .	9-30
uitree Function: Create trees with editable node text in the running app . . . . .	9-30
Component Text Alignment: Improved text alignment for labels, check boxes, and radio buttons . . . . .	9-30
Functionality being removed or changed . . . . .	9-31
<b>Performance</b> . . . . .	<b>9-32</b>
Startup: Increased speed of MATLAB startup time . . . . .	9-32
Execution Engine: Execute tight loops with scalar math faster . . . . .	9-32

Execution Engine: Improved performance for common programming patterns . . . . .	9-32
App Designer: Starting, loading, and layout tasks are faster . . . . .	9-32
<b>Hardware Support . . . . .</b>	<b>9-33</b>
Raspberry Pi: Support for Raspberry Pi Zero W board . . . . .	9-33
MATLAB Online: Acquire live images from USB webcams in MATLAB Online . . . . .	9-33
<b>Advanced Software Development . . . . .</b>	<b>9-34</b>
Tab Completion: Describe your function syntaxes for custom tab completion and other contextual suggestions . . . . .	9-34
Unit Testing Framework: Run tests from the MATLAB Editor toolstrip . . . . .	9-34
App Testing Framework: Author automated tests for App Designer apps . . . . .	9-34
Unit Testing Framework: Rerun failed tests with one click . . . . .	9-34
Unit Testing Framework: Test if values point to existing files or folders with IsFile and IsFolder constraints . . . . .	9-34
Unit Testing Framework: Test if two sets are the same with IsSameSetAs constraint . . . . .	9-35
Unit Testing Framework: Select tests by test class hierarchy . . . . .	9-35
Unit Testing Framework: Direct output stream to unique files for plugins . . . . .	9-35
Unit Testing Framework: Increased access to parameterized testing properties . . . . .	9-36
Unit Testing Framework: Compare cell arrays of character arrays using StringComparator . . . . .	9-36
Unit Testing Framework: Comparison method for objects changed . . . . .	9-37
Performance Testing Framework: Define multiple, labeled measurement boundaries in test methods . . . . .	9-37
Mocking Framework: Specify default property values on mock object . . . . .	9-37
Mocking Framework: Obtain interaction history for mock object . . . . .	9-37
Mocking Framework: Construct mocks for classes that have Abstract properties with other attributes . . . . .	9-37
matlab.net.http Package: Stream data to and from a web service and handle forms and multipart messages . . . . .	9-38
C++ MEX Interface: Access MATLAB data and objects easier from C++ . . . . .	9-38
Class Constructors: Author subclass without implementing a constructor solely to pass arguments through to a superclass constructor . . . . .	9-38
Property Validation: Get information about property validation . . . . .	9-38
Property Validation: Define validation for abstract properties . . . . .	9-39
Functions: Call numArgumentsFromSubscript for object dot method from overloaded subsref . . . . .	9-39
Classes: Concatenate matlab.lang.OnOffSwitchState enumeration members with nonmember char and string . . . . .	9-39
Python Version 3.4: Support discontinued . . . . .	9-40
Source Control Integration: View changes, save revisions, and manage repository locks . . . . .	9-40
MATLAB Engine API for C++: Set and get a property value on an object in an object array . . . . .	9-40
MATLAB Data API: Applications built with R2018a API do not run in MATLAB R2017b . . . . .	9-40
MEX Functions: Build C MEX Files with Interleaved Complex API . . . . .	9-40

MEX Functions: Release-specific build options .....	9-41
Version Embedded in MEX Files .....	9-41
Perl 5.26.1: MATLAB support .....	9-41
System objects: Create System Objects in MATLAB .....	9-42
System object support for strings .....	9-42
.NET: Supports string data type .....	9-42
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications .....	9-43
Functionality being removed or changed .....	9-43

## R2017b

<b>Desktop .....</b>	<b>10-2</b>
Live Editor: Write MATLAB commands with automated, contextual hints for arguments, property values, and alternative syntaxes .....	10-2
Live Editor: Export live scripts to LaTeX format .....	10-2
Live Editor: Display high-resolution plots in PDF output .....	10-2
Live Editor: Horizontally align text, equations, and images .....	10-2
Live Editor: Automatically match delimiters and wrap comments while editing code .....	10-2
Live Editor: View and scroll through table data, including variable and row names .....	10-3
Live Editor: Check code for errors and warnings using the message bar and message indicator .....	10-3
Documentation: Use the Live Editor in a web browser to open, edit, and run MATLAB online documentation examples .....	10-3
MATLAB Drive: Store, access, and manage your files from anywhere ...	10-3
Add-On Manager: Customize your MATLAB environment by enabling and disabling add-ons .....	10-4
Add-On Manager: Find installed add-ons faster using sort and search ...	10-4
Toolbox Packaging: Create a Getting Started Guide for your toolbox from a Live Script template .....	10-4
Toolbox Packaging: Share your toolbox on File Exchange directly when you package it .....	10-4
Command Window: View updated display for cell arrays .....	10-4
<b>Language and Programming .....</b>	<b>10-6</b>
Code Compatibility Report: Generate a report that helps the updating of code to a newer MATLAB release .....	10-6
isStringScalar Function: Determine whether input is a string array with one element .....	10-6
convertStringsToChars and convertCharsToStrings Functions: Enable your code to accept all text types as inputs without otherwise altering your code .....	10-6
arrayfun, cellfun, and structfun Functions: Return object arrays as output arguments .....	10-6
Scripts: Run sections in scripts containing local functions .....	10-6
isfile and isfolder Functions: Determine if input is a file or a folder .....	10-7
Functionality being removed or changed .....	10-7

<b>Mathematics</b> .....	<b>10-13</b>
decomposition Object: Solve linear systems repeatedly with improved performance .....	<b>10-13</b>
lsqminnorm Function: Find minimum-norm solution of underdetermined linear system .....	<b>10-13</b>
dissect Function: Reorder sparse matrix columns using nested dissection ordering .....	<b>10-13</b>
vecnorm Function: Compute vector-wise norms of arrays .....	<b>10-13</b>
polyshape Object: Create, analyze, and visualize 2-D polygons .....	<b>10-13</b>
eigs Function: Improved algorithm and new options .....	<b>10-13</b>
svds Function: Set options with name-value pairs .....	<b>10-15</b>
Interpolation Functions: Method for modified Akima cubic Hermite interpolation .....	<b>10-16</b>
convn Function: Compute convolutions on multidimensional arrays with improved performance .....	<b>10-16</b>
subgraph and highlight Functions: Specify graph nodes with logical vector .....	<b>10-16</b>
Functionality being removed or changed .....	<b>10-17</b>
<b>Graphics</b> .....	<b>10-18</b>
geobubble Function: Create an interactive map with bubbles whose size and color vary with data values .....	<b>10-18</b>
wordcloud Function: Display words at different sizes based on frequency or custom size data .....	<b>10-18</b>
binscatter Function: Visualize data density with dynamic bin size adjustment .....	<b>10-18</b>
Tall Array Support: Visualize out-of-memory data using plot, scatter, and binscatter .....	<b>10-18</b>
heatmap Function: Sort rows and columns and use custom labels in a heatmap .....	<b>10-18</b>
bar Function: Control individual bar colors .....	<b>10-19</b>
Chart Colors: Create bar and area charts with new default colors .....	<b>10-19</b>
Axes Object: Specify the target axes for more functions .....	<b>10-20</b>
Functionality being removed or changed .....	<b>10-21</b>
<b>Data Import and Export</b> .....	<b>10-24</b>
Custom Datastore: Build a customized datastore .....	<b>10-24</b>
datastore Function: Work with data stored in Windows Azure Blob Storage .....	<b>10-24</b>
datastore Function: Access Hadoop HDFS data more easily .....	<b>10-24</b>
FileDatastore Object: Create uniform output from datastore .....	<b>10-24</b>
HDF5 Functions: Create datasets, groups, attributes, links, and named datatypes using non-ASCII characters .....	<b>10-24</b>
Web services: Skip server name verification in certificates .....	<b>10-25</b>
jsonencode Function: Encode NaN and Inf as null .....	<b>10-25</b>
Functionality being removed or changed .....	<b>10-26</b>
<b>Data Analysis</b> .....	<b>10-29</b>
ischange Function: Detect abrupt changes in data .....	<b>10-29</b>
islocalmin and islocalmax Functions: Detect local minima and maxima in data .....	<b>10-29</b>
rescale Function: Scale data to a specified range .....	<b>10-29</b>

tall Arrays: Operate on tall arrays with more functions, including fillmissing, filter, median, polyfit, and synchronize . . . . .	10-29
tall Array Indexing: Use subscripted assignment with tall arrays . . . . .	10-29
tallrng Function: Control random number generator used by tall arrays . . . . .	10-30
timetable Data Container: Specify whether each variable in a timetable contains continuous or discrete data using the VariableContinuity property . . . . .	10-30
mink and maxk Functions: Find the k smallest or largest elements in an array . . . . .	10-30
topkrows Function: Find the k top rows in sorted order for numeric arrays, tables, and timetables . . . . .	10-30
<b>App Building . . . . .</b>	<b>10-31</b>
App Designer: Create apps with a wide variety of 2-D and 3-D plots . . .	10-31
App Designer: Add menus to an app from the Component Library . . . . .	10-31
App Designer: Specify input arguments when running an app . . . . .	10-31
App Designer: Add a summary, description, and screenshot for app packaging and compiling . . . . .	10-31
App Designer: Improved component Properties pane in Code View . . . . .	10-31
App Designer: Edit tick labels for gauges, knobs, and sliders directly in the canvas . . . . .	10-31
uitree and uitreenode Functions: Create trees and tree nodes in apps . .	10-31
uiconfirm Function: Create modal in-app confirmation dialog boxes . . .	10-32
Toolbox Packaging: Add App Designer apps to the Apps Gallery upon toolbox installation . . . . .	10-32
MATLAB Online: Run App Designer apps in MATLAB Online . . . . .	10-32
<b>Performance . . . . .</b>	<b>10-33</b>
App Designer: Load apps faster . . . . .	10-33
Execution Engine: Improved performance for vectorized math on CPUs with AVX2 . . . . .	10-33
Live Editor: Run live scripts with loops faster . . . . .	10-33
<b>Hardware Support . . . . .</b>	<b>10-34</b>
Arduino: Wirelessly connect to Arduino boards using low-cost Bluetooth adaptors . . . . .	10-34
Arduino Setup UI: Set up a connection to your Arduino board over USB, Bluetooth, or Wi-Fi . . . . .	10-34
Arduino Plug-In Detection: Discover available Arduino support and examples when plugging a compatible Arduino board . . . . .	10-34
<b>Advanced Software Development . . . . .</b>	<b>10-35</b>
MATLAB Engine API for C++: Run MATLAB code from C++ programs with object-oriented programming support and asynchronous execution . .	10-35
MATLAB Engine API for C++: Pass data between C++ programs and MATLAB using MATLAB Data Array . . . . .	10-35
Java SE 8: MATLAB support, providing improved security and access to new Java features . . . . .	10-35
MinGW 5.3: MATLAB support . . . . .	10-35
Microsoft Visual Studio 2017: MATLAB support for Microsoft Visual Studio 2017 Community, Professional, and Enterprise editions . . . . .	10-35

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications . . . . .	10-35
Python Version 3.6: MATLAB support . . . . .	10-36
Perl 5.24.1: MATLAB support . . . . .	10-36
MATLAB Handle class method: Add a listener for an event without binding the listener to the source object . . . . .	10-37
Unit Testing Framework: Provide code coverage reports in the Cobertura format for improved continuous integration workflows . . . . .	10-37
Unit Testing Framework: Generate HTML report of a test run . . . . .	10-37
Unit Testing Framework: Write tests as live scripts . . . . .	10-37
Unit Testing Framework: Specify additional diagnostics to evaluate upon failures using the onFailure method . . . . .	10-37
Performance Testing Framework: Define multiple measurement boundaries in test methods . . . . .	10-37
Mocking Framework: Construct mocks for classes that have Abstract methods with other attributes . . . . .	10-37
Source Control Integration: Show differences from parent files and save copies in Git Branches . . . . .	10-38
Functionality being removed or changed . . . . .	10-38

## R2017a

<b>Desktop . . . . .</b>	<b>11-2</b>
Live Editor: Edit a figure interactively including title, labels, legend, and other annotations . . . . .	11-2
Live Editor: Get suggestions for mistyped commands and variables . . . . .	11-2
Live Editor: Copy live script outputs to other applications . . . . .	11-2
Live Editor: Hover over variables to see their current value . . . . .	11-2
Add-On Explorer: Discover and install File Exchange submissions hosted on GitHub in Add-On Explorer . . . . .	11-3
MATLAB Online: Use MATLAB through your web browser for teaching, learning, and convenient, lightweight access . . . . .	11-3
Startup Folder Behavior Changes: Set initial working folder using new options and behaviors . . . . .	11-3
<b>Language and Programming . . . . .</b>	<b>11-4</b>
string Arrays: Create string arrays using double quotes . . . . .	11-4
String Functions: Return character arrays or cell arrays instead of string arrays . . . . .	11-4
missing Function: Assign missing values in core data types, including double, datetime, categorical, and string arrays . . . . .	11-4
issortedrows Function: Determine if matrix and table rows are sorted . . . . .	11-4
sort and sortrows Functions: Specify options for sorting complex numbers and placing missing elements . . . . .	11-4
issorted Function: Query sort order with monotonic, strictly monotonic, strictly ascending, and strictly descending options . . . . .	11-5
head and tail Functions: Return top or bottom rows of table or timetable . . . . .	11-5
table Data Containers: Use row labels when performing join, sort, and grouping operations . . . . .	11-5



Functionality being removed or changed . . . . .	11-5
<b>Graphics . . . . .</b>	<b>11-8</b>
heatmap Function: Visualize table or matrix data as a heatmap . . . . .	11-8
legend Function: Create legends that update when data is added to or removed from the axes . . . . .	11-8
Categorical Plotting: Use categorical data in common plotting functions and customize axes with categorical rulers . . . . .	11-10
histogram Function: Plot histograms of datetime and duration data . . .	11-10
histogram Function: Sort categorical bins by bar height, and limit the number of bins displayed . . . . .	11-10
scatter Function: Create scatter plots of datetime and duration data . .	11-11
Scatter Plots: Create scatter plots with varying marker sizes faster . . .	11-11
parula Colormap: Create plots with enhanced colors . . . . .	11-11
Functionality being removed or changed . . . . .	11-11
<b>Data Import and Export . . . . .</b>	<b>11-14</b>
datastore and tabularTextDatastore Functions: Automatically detect and return date and time data in text files . . . . .	11-14
datastore Function: Work with data in Amazon S3 cloud storage . . . . .	11-14
Import Tool: Import strings and categorical arrays interactively . . . . .	11-14
detectImportOptions Function: Control import properties of fixed-width text files . . . . .	11-14
RESTful web services: Support for PUT and DELETE HTTP methods in webread, webwrite, and websave . . . . .	11-15
save Function: Save workspace variables to a MAT-file with or without compression . . . . .	11-15
writetable Function: Select preferred character encoding when writing to a file . . . . .	11-15
NetCDF Functions: Create variable names and attributes containing non- ASCII characters . . . . .	11-15
Webcam Support Package: GStreamer Upgrade on Linux . . . . .	11-15
jsondecode converts JSON null values in numeric arrays to NaN . . . . .	11-15
load and fopen Functions: Use the file separator character ('\') preceding a file name to indicate that the file is in the root folder . . . . .	11-16
Functionality being removed or changed . . . . .	11-17
<b>Data Analysis . . . . .</b>	<b>11-18</b>
tall Arrays: Operate on tall arrays with more functions, including ismember, sort, conv, and moving statistics functions . . . . .	11-18
tall Arrays: Index tall arrays using sorted indices . . . . .	11-18
tall Arrays: Work with out-of-memory, time-stamped data in a timetable . . . . .	11-18
isoutlier and filloutliers Functions: Detect and replace outliers in an array or table . . . . .	11-18
smoothdata Function: Smooth noisy data in an array or table with filtering or local regression . . . . .	11-19
summary Function: Calculate summary statistics and variable information in tables and timetables . . . . .	11-19
histcounts Function: Bin datetime and duration data . . . . .	11-19
movmad and movprod Functions: Compute moving median absolute deviation and moving product of an array . . . . .	11-19

bounds Function: Simultaneously determine the smallest and largest elements of an array . . . . .	11-19
fillmissing Function: Replace missing data in an array or table using moving mean or moving median option . . . . .	11-19
Moving Statistics Functions: Supply sample points for time-stamped and nonuniform data in moving statistics functions, such as movmean . . . . .	11-19
prod and cumprod Functions: Ignore NaNs using 'omitnan' . . . . .	11-19
Functionality being removed or changed . . . . .	11-20
<b>App Building . . . . .</b>	<b>11-21</b>
App Designer: Learn to build apps using an interactive tutorial . . . . .	11-21
App Designer: Zoom and pan plots . . . . .	11-21
App Designer: Configure columns of a table to automatically fill the entire width of the table . . . . .	11-21
App Designer: Manage common design-time settings using the Preferences dialog box . . . . .	11-21
App Designer: Include comet, graph, and digraph visualizations in apps . . . . .	11-21
App Designer: Write ButtonDownFcn callbacks for graphics objects displayed in UI axes . . . . .	11-21
App Designer: Edit table column headings directly in the canvas . . . . .	11-22
App Designer: Disable automatic resize behavior when writing SizeChangedFcn callbacks . . . . .	11-22
<b>Performance . . . . .</b>	<b>11-23</b>
Execution Engine: Improved performance for setting MATLAB object properties . . . . .	11-23
save Function: Save MAT v7.3 files without compression for improved performance on some storage devices . . . . .	11-23
memoize Function: Cache results of a function to avoid rerunning when called with the same inputs . . . . .	11-23
Scripts: Improved performance of scripts with lower script overhead . . . . .	11-23
try, catch Block: Improved performance of try blocks with lower execution overhead . . . . .	11-23
App Designer: Load apps faster . . . . .	11-23
Mathematics Functions: Various performance improvements . . . . .	11-23
<b>Hardware Support . . . . .</b>	<b>11-25</b>
Arduino: Read from quadrature encoders . . . . .	11-25
Arduino: Wirelessly connect to Arduino MKR1000 board over Wi-Fi . . . . .	11-25
<b>Advanced Software Development . . . . .</b>	<b>11-26</b>
Class matlab.lang.OnOffSwitchState: Represent on and off as logical values . . . . .	11-26
Object Properties: Validate object property values by their type, size, shape, or other parameters . . . . .	11-26
Validation Functions: Validate that values meet specific criteria by calling the appropriate function . . . . .	11-26
Mocking Framework: Isolate a portion of a system to test by imitating behavior of dependent components . . . . .	11-27
Unit Testing Framework: Generate screenshots and figures during testing with ScreenshotDiagnostic and FigureDiagnostic . . . . .	11-27

Unit Testing Framework: Capture screenshots and figures generated during tests using TestReportPlugin .....	<b>11-27</b>
Unit Testing Framework: Control runtests function with debug, strict, and verbosity options .....	<b>11-28</b>
Unit Testing Framework: Select tests by procedure name .....	<b>11-28</b>
Unit Testing Framework: Comparator for MATLAB tables .....	<b>11-28</b>
Performance Testing Framework: View statistics from test measurements with the sampleSummary method .....	<b>11-28</b>
Performance Testing Framework: Apply a function across test measurements with the samplefun method .....	<b>11-29</b>
Source Control Integration: Use Git Pull to fetch and merge in one step .....	<b>11-29</b>
MEX builds with 64-Bit API by default .....	<b>11-29</b>
MEX files and shared libraries: Diagnostic information displayed for failure to load .....	<b>11-30</b>
Java: Supports string data type .....	<b>11-30</b>
Python: Supports string data type .....	<b>11-30</b>
Python Version 3.3: Support discontinued .....	<b>11-31</b>
MATLAB ships with ActiveState Perl version 5.24 on Windows platforms .....	<b>11-31</b>
Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications .....	<b>11-31</b>
Functionality being removed or changed .....	<b>11-31</b>



# R2022a

---

**Version: 9.12**

**New Features**


**Bug Fixes**

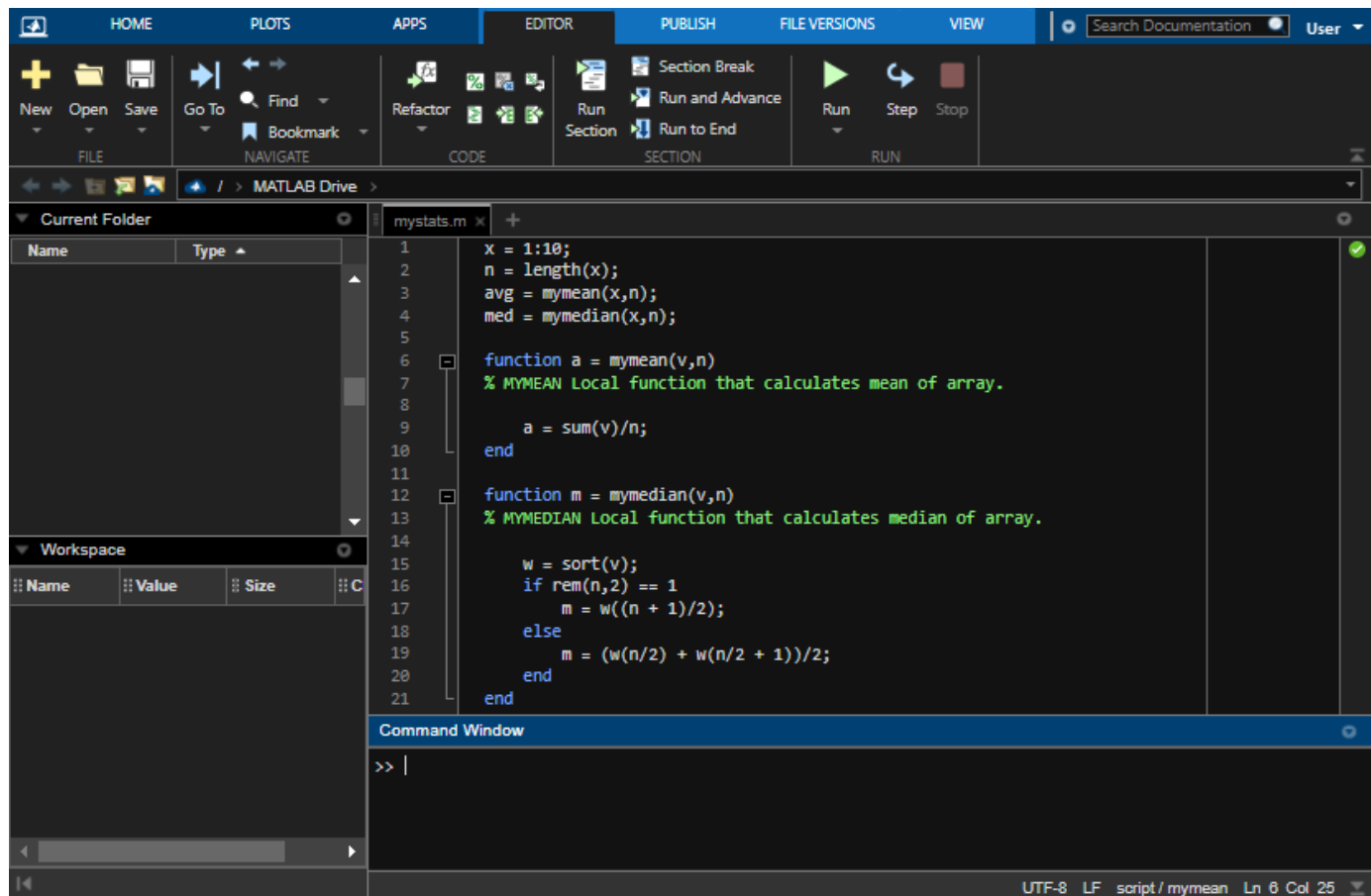
**Version History**

## Environment

### Themes in MATLAB Online: Change the colors of the MATLAB desktop by selecting a dark or light theme

In MATLAB Online™, you can change the colors of the MATLAB desktop using themes.

For example, to select a dark theme, on the **Home** tab, in the **Environment** section, click  **Preferences**. Select **MATLAB > Appearance** and set the **Theme** to **Dark**.



To further customize the colors of the MATLAB desktop, select **MATLAB > Appearance > Colors**. Then, change the colors in the **Desktop tool colors**, **MATLAB syntax highlighting colors**, and **MATLAB output colors** sections.

As part of this change, icons in the MATLAB Online desktop have an improved visual appearance.

For more information about changing the colors of the MATLAB desktop, see “Change Desktop Colors”.

## Live Editor Colors: Change the text and background colors of live scripts and functions

You can change the text and background colors in the Live Editor by changing the MATLAB desktop tool colors.

To change the text and background colors:

- 1 On the **Home** tab, in the **Environment** section, click  **Preferences**.
- 2 Select **MATLAB > Colors**

In MATLAB Online, select **MATLAB > Appearance > Colors**.


- 3 In the **Desktop tool colors** section, clear the **Use system colors** check box.

In MATLAB Online, the **Use system colors** check box is not available and this step can be skipped.

- 4 Use the **Text** and **Background** fields to change the colors. For example, select white for the text color and black for the background color.

The Live Editor automatically selects colors for titles and headings based on the selected background color. To further customize the colors of titles and headings, use settings. For more information, see `matlab.fonts` Settings.

## Live Editor Hyperlinks: Insert hyperlinks to specific locations in separate live scripts or live functions

Use hyperlinks to navigate to a location in a separate, existing live script or function. To insert a hyperlink, select the text to link in the current file, go to the **Insert** tab, and click  **Hyperlink**. Edit your display text (optional), select **Location in existing document**, and enter or browse for the file path. Then, select a location in the document preview that displays on the right.

For more information about inserting hyperlinks, see “Format Text in the Live Editor”.

## Live Editor Export: Export live scripts and functions programmatically using the export function

Use the `export` function to programmatically export live scripts and functions to a standard format. Available formats include PDF, Microsoft® Word, HTML, and LaTeX.

For example, to export the live script `homework1.mlx` as HTML, type:

```
export("homework1.mlx","homework1.html")
```


## Live Editor Accessibility: Interact with output in live scripts using the keyboard

You can now use keyboard shortcuts to interact with output in live scripts when output is on the right. To move focus from the code to the output display panel, press **Ctrl+Shift+O**. On macOS, press **Option+Command+O**. To activate an output, press **Enter**. Once an output is activated, you can

scroll text using the arrow keys, navigate through hyperlinks and buttons using the **Tab** key, and open the context menu by pressing **Shift+F10**.

## Live Editor Tasks: View and interact with tasks when code is hidden

When you hide code in a live script, the Live Editor now displays Live Editor tasks along with formatted text, labeled controls, and output.

To hide code, select the Hide Code button  to the right of the live script or in the **View** tab. Alternatively, if you are using the export function, you can hide the code using the `HideCode` name-value argument.

If a Live Editor task is configured to show only code and no controls, then the task does not display when you hide code.

## Component Browser: Reorder children in App Designer or the Property Inspector



You can now drag one or more Axes object children, Group object children, or Transform object children sharing the same parent to reorder them in the Component Browser in App Designer or the Property Inspector for figures.

When reordering children, use visual feedback for allowed moves. You can undo and redo the reordering of the children with **Undo** or **Redo** or the corresponding keyboard shortcuts.

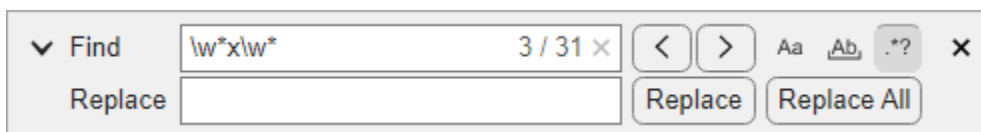
## Editor Python Support: View and edit Python files with syntax highlighting, auto-indenting, and delimiter matching

The Editor now displays Python® files with syntax highlighting for keywords, strings, comments, and errors. In addition, the Editor auto-indent Python files and indicates matched and mismatched delimiters such as parentheses, brackets, and braces.

## Find and Replace Dialog Box: Search text in the Editor and Live Editor using regular expressions

You can use a subset of regular expressions to search for text that matches a pattern in an open file in the Editor or Live Editor. To search using a regular expression, on the **Editor** or **Live Editor** tab, in the **Navigate** section, click  Find. Then, in the find and replace dialog box, enter a regular expression, and select the Regular Expression button .

For example, to find all the words in a file that contain the letter x, enter the expression `\w*x\w*` and select the Regular Expression button .



For more information, see “Find and Replace Text in Files and Go to Location”.



## Profiler: Access the Profiler from the Apps tab

The MATLAB Profiler is now available as an app and can be found in the MATLAB section of the apps gallery in the **Apps** tab.

You can still access the Profiler from the **Home** tab, in the **Code** section, by clicking the **Run and Time** button, or programmatically using the `profile` function.

## Internationalization: UTF-8 system encoding on Windows platforms

MATLAB now uses UTF-8 as its system encoding on Windows®, completing the adoption of Unicode® across all supported platforms. MATLAB has used UTF-8 as the default encoding for MATLAB files and file I/O since R2020a.

## Installation Settings: Configure persistent settings for MATLAB installations

Installation-level settings provide a new layer of MATLAB configuration that lies between factory settings and personal settings. Installation settings override the factory settings for all users of a given MATLAB installation, and they are persistent across sessions.

In previous versions of MATLAB, if administrators wanted to limit RAM usage for a set of MATLAB users by lowering the `ArraySizeLimit`, they had to create and distribute a script to change the personal setting for each individual user. Starting in R2022a, the administrator can apply the change to all users of their MATLAB installation using the installation setting for `ArraySizeLimit`.

Access installation-level settings using the new `InstallationValue` property of the `Setting` object. Verify and clear installation settings with two new object functions, `hasInstallationValue` and `clearInstallationValue`, respectively.

## Comparison Tool: Save results as HTML report

You can now use the Comparison Tool to publish text comparison results in an HTML report. For more details, see “Compare Text Files”.

## Comparison Tool: Compare folders in MATLAB Online

Starting in R2022a, you can compare folders and zip files in MATLAB Online.

You can access the comparison tool from:

- The MATLAB Current Folder browser context menu
- The Current Project browser context menu
- The MATLAB Command Window using the `visdiff` function

## Functionality being removed or changed

### Live Editor figure size is bounded upon saving

*Behavior change*

When opening a saved live script, existing images in the output have a maximum size equivalent to the figure size upon saving. To adjust the size past this maximum limit, you can run the live script and increase the figure size.

## Language and Programming

### **Class Introspection: Description and DetailedDescription properties of metaclasses contain text from code comments**

The `Description` and `DetailedDescription` properties of these metaclasses pull content from code comments:

- `meta.class`
- `meta.method`
- `meta.property`
- `meta.event`
- `meta.EnumeratedValue`

For user-defined classes with appropriately placed code comments, the `Description` and `DetailedDescription` properties of the metaclasses are populated with text pulled from those comments. For more information on how to use code comments to store custom help text for user-defined classes, see “Custom Help Text”.

### **Class Introspection: Access class aliases from meta.class instance**

The aliases of a class are stored in the new `Aliases` property of `meta.class`. For more information on class aliasing, see “Creating and Managing Class Aliases”.

### **Background Pool: See futures in the background**

Starting in R2022a, you can query all queued and running futures in the background by using the `FevalQueue` property of the pool. To create futures, use `parfeval` and `parfevalOnAll`. For more information on futures, see `Future`.

### **cancelAll Method: Cancel currently queued and running futures in the background pool**

`cancelAll` cancels all futures currently queued or running in the background pool. Queued or running futures are listed in the `FevalQueue` property.

### **Background Pool: Check the status of the background pool**

Starting in R2022a, you can query to determine if the background pool is currently running by using the `Busy` property of the pool. This property indicates whether the background pool is busy, specified as `true` or `false`. The pool is busy if there is outstanding work for the pool to complete.

### **pcode Function: Create P-code files with enhanced obfuscation**

The `pcode` function now has the option “-R2022a”, which creates P-code files using a more complex obfuscation algorithm. Files created with this option run only in MATLAB releases R2022a and later.

## **str2num Function: Restrict evaluation to basic math expressions**

`str2num` is implemented using the `eval` function, which evaluates the input argument. Starting in R2022a, you can set the name-value argument `Evaluation` to "restricted" to restrict accepted inputs to basic math expressions, such as `200` and `1+2i`.

## **Functionality being removed or changed**

### **Defining classes and packages: Using `schema.m` will not be supported in a future release** *Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

### **dec2hex and dec2bin input types are now restricted**

*Behavior change*

Input types allowed by `dec2hex` and `dec2bin` have been restricted. Supported input types are primitive numeric types and classes that inherit from a primitive numeric type.

In addition, `dec2bin(0,0)` will now return `'0'` rather than a `1x0` character vector.

### **cd no longer removes leading spaces for Windows drive letter paths**

*Behavior change*

Before R2022a, on Windows platforms, the `cd` function removed leading spaces in input paths specifying the drive letter. Input paths containing leading spaces now cause an error to be thrown instead. If an input path is invalid with leading spaces, then use `strip` to remove the spaces before using the `cd` function.

### **TruncateScalarObject name-value argument for widthConstrainedDataRepresentation method renamed to AllowTruncatedDisplayForScalar**

*Behavior change in future release*

The name of the `TruncateScalarObject` name-value argument for the `widthConstrainedDataRepresentation` method is now `AllowTruncatedDisplayForScalar`. The functionality of the option will not change. Support for the name `TruncateScalarObject` will be removed in a future release.

### **cast returns consistent output for subclass of MATLAB numeric types**

*Behavior change*

The syntax `cast(A, "like", p)` now returns output consistent with the prototype `p` when the data type of `p` is a subclass of MATLAB numeric types.

For example, this code returns an output that has the same data type as `p`:

```
p = matlab.lang.OnOffSwitchState.on;  
x = cast(1, "like", p)
```

```
x =
```

```
OnOffSwitchState enumeration
```

```
on
```

In previous releases, the code returns `x = 1` with data type `logical`.

## Data Analysis

### Data Cleaner App: Interactively preprocess and organize column-oriented data

The new **Data Cleaner** app enables you to:

- Access column-oriented data in the MATLAB workspace or import column-oriented data from a file.
- Explore data by using the visualization, data, and summary views.
- Sort by a variable, rename a variable, or remove a variable.
- Retime data in a timetable, stack or unstack table variables, clean missing data, clean outlier data, smooth data, or normalize data.
- Edit previously performed cleaning steps by using the **Cleaning Steps** panel.
- Export cleaned data to the MATLAB workspace, or export code for cleaning data as a script or function.

You can open the Data Cleaner app from the MATLAB section of the apps gallery in the **Apps** tab. Alternatively, enter `dataCleaner` in the MATLAB command window.

The **Data Cleaner** app currently supports cleaning only timetable data and importing only one timetable at a time.

### **allfinite, anynan, and anymissing Functions: Determine if all array elements are finite, any element is NaN, and any element is missing**

Use the `allfinite`, `anynan`, and `anymissing` functions to examine the elements of an input array.

- `allfinite`: Determine if all array elements are finite.
- `anynan`: Determine if any array element is NaN.
- `anymissing`: Determine if any array element is missing.

### **quantile, prctile, and iqr Functions: Calculate quantiles, percentiles, and interquartile range**

Calculate quantiles, percentiles, and the interquartile range of a data set by using the `quantile`, `prctile`, and `iqr` functions.

Previously, the `quantile`, `prctile`, and `iqr` functions required Statistics and Machine Learning Toolbox™.

### **rms Function: Calculate root-mean-square value**

Calculate the root-mean-square (RMS) value of input data with `rms`.

You can specify the dimensions to operate along and whether to include or omit NaN values in the calculation:

- Use "all" to calculate the RMS value of all elements of the input array.
- Use the `dim` input argument to calculate the RMS value along one dimension.
- Use the `vecdim` input argument to calculate the RMS value along multiple dimensions.
- Use "includenan" or "omitnan" to include or omit NaN values in the RMS calculation.

Previously, the `rms` function required Signal Processing Toolbox™.

## std and var Functions: Optionally return mean as a second output

The `std` and `var` functions can now return the mean of the elements used to calculate the standard deviation or variance by using a second output argument `M`. If a weighting scheme is specified, then the weighted mean is returned.

## Date and Time Functions: Some Financial Toolbox functions combined with MATLAB functions

The following date and time functions from Financial Toolbox™ are combined with functions having the same names in MATLAB. Before R2022a, these Financial Toolbox functions supported serial date numbers and text timestamps as inputs, while the MATLAB functions supported `datetime` arrays. Starting in R2022a, the MATLAB functions support `datetime` arrays, serial date numbers, and text timestamps as inputs. The functions are removed from Financial Toolbox.

- `day`
- `hour`
- `minute`
- `month`
- `quarter`
- `second`
- `year`

## Version History

While these functions support serial date number and text inputs, these types of inputs are not recommended. Use `datetime` values as inputs instead. The `datetime` data type provides flexible date and time formats, storage out to nanosecond precision, and properties to account for time zones and daylight saving time. To convert serial date numbers or text timestamps to `datetime` values, use the `datetime` function.

- To convert serial date numbers to `datetime` values, call `datetime` with the `ConvertFrom` name-value argument:

```
dt = datetime(738457, "ConvertFrom", "datenum")
```

```
dt =
```

```
datetime
```

```
28-Oct-2021
```

- To convert text timestamps, call `datetime`.

```
dt = datetime("2021-10-28")
```

```
dt =
```

```
    datetime
```

```
    28-Oct-2021
```

There are no plans to remove support for serial date numbers or text timestamps from these MATLAB functions.

## Date and Time Functions: Some Financial Toolbox functions moved to MATLAB

These date and time functions are removed from Financial Toolbox and moved to MATLAB:

- `eomdate`
- `lweekdate`
- `m2xdate`
- `months`
- `nweekdate`
- `today`
- `weeknum`
- `x2mdate`

## Version History

While MATLAB supports these functions, they are not recommended because they use serial date numbers in their calculations. The table shows recommended replacements that either accept or return `datetime` values. The `datetime` data type is recommended because it provides flexible date and time formats, storage out to nanosecond precision, and properties to account for time zones and daylight saving time.

There are no plans to remove these functions from MATLAB.

Transferred Function	Recommended Replacement
<code>eomdate</code>	<code>dateshift</code> , with <code>datetime</code> values as inputs
<code>lweekdate</code>	<code>lweekdate</code> , with <code>outputType</code> specified as "datetime" to return <code>datetime</code> output
<code>m2xdate</code>	<code>exceltime</code> , with <code>datetime</code> values as inputs
<code>months</code>	<code>between</code> , with <code>datetime</code> values as inputs
<code>nweekdate</code>	<code>nweekdate</code> , with <code>outputType</code> specified as "datetime" to return <code>datetime</code> output
<code>today</code>	<code>datetime</code> , with "today" as the input argument
<code>weeknum</code>	<code>week</code> , with <code>datetime</code> values as inputs



Transferred Function	Recommended Replacement
x2mdate	datetime, with dateType specified as "excel"

## matlab.datetime.compatibility.convertDatenum Function: Convert text timestamps and serial date numbers to datetime values in a backward-compatible way

To convert text timestamps and serial date numbers to `datetime` values, use the `matlab.datetime.compatibility.convertDatenum` function. For backward compatibility, this function supports the subset of `datestr` formats that the `datenum` function recognizes when it converts text timestamps without a format specifier.

Use this function in code where you intend to return `datetime` values, but to preserve compatibility you need to interpret text inputs in the same way that `datenum` interprets them. This function is designed to be a compatibility layer for function authors.

To explicitly convert serial date numbers to `datetime` values, use the `datetime` function instead, with the `ConvertFrom` name-value argument:

```
dt = datetime(738457, "ConvertFrom", "datenum")
dt =
    datetime
    28-Oct-2021
```

## categorical Data Type: Use a pattern object to specify category names that match a pattern

When you specify category names of a `categorical` array, you can use a `pattern` object to specify names that match a pattern.

For example, suppose you have a `categorical` array that has many different categories that can represent "yes" and "no". This `categorical` array has six values and six categories because the values in the input array are different.

```
C = categorical(["Y" "Yes" "Yeah" "N" "No" "Nope"])
C =
    1×6 categorical array
    Y      Yes      Yeah      N      No      Nope
```

To combine all the different "yes" categories into one category and all the different "no" categories into another category, use the `mergecats` function and wildcard patterns to match the category names. The `categorical` array still has six values. But it has only two categories, "yes" and "no".

```
C = mergecats(C, "Y" + wildcardPattern, "yes");
C = mergecats(C, "N" + wildcardPattern, "no")
C =
```

```
1×6 categorical array
```

```
yes     yes     yes     no     no     no
```

These functions provide support for using patterns when you specify category names:

- `histcounts` (when you specify the `Categories` argument)
- `iscategory` (when you specify the `catnames` argument)
- `mergecat`s (when you specify the `oldcats` argument)
- `removecats` (when you specify the `oldcats` argument)
- `reordercats` (when you specify the `neworder` argument)

## table and timetable Data Types: Use a pattern object to specify row, variable, and property names that match a pattern

When you specify rows, variables, or properties of a table or timetable, you can use a pattern object to specify names that match a pattern.

You can use patterns when you subscript into a table by row names and variable names, or when you subscript into a timetable by variable names.

For example, read a table into MATLAB.

```
T = readtable("outages.csv","TextType","string")
```

```
T =
```

```
1468×6 table
```

Region	OutageTime	Loss	Customers	RestorationTime	Cause
"SouthWest"	2002-02-01 12:18	458.98	1.8202e+06	2002-02-07 16:50	"winter storm"
"SouthEast"	2003-01-23 00:49	530.14	2.1204e+05	NaT	"winter storm"
"SouthEast"	2003-02-07 21:15	289.4	1.4294e+05	2003-02-17 08:14	"winter storm"
:	:	:	:	:	:

To subscript into the table and select all variables whose names end with "Time", use a wildcard pattern.

```
T2 = T(:,wildcardPattern + "Time")
```

```
T2 =
```

```
1468×2 table
```

OutageTime	RestorationTime
2002-02-01 12:18	2002-02-07 16:50
2003-01-23 00:49	NaT
2003-02-07 21:15	2003-02-17 08:14
:	:

These functions provide support for using patterns when you specify variables by name:

- `convertvars`
- `innerjoin`
- `issortedrows`
- `join`
- `movevars`
- `mergevars`
- `outerjoin`
- `removevars`
- `rowfun`
- `rows2vars`
- `sortrows`
- `splitvars`
- `stack`
- `topkrows`
- `unstack`
- `varfun`

This function provides support for using patterns when you specify properties by name:

- `rmprop`

### **Data Preprocessing Functions: Append transformed variables to input data using the `ReplaceValues` name-value argument**

When you preprocess tables and timetables, you can now append variables containing the transformed values to the input table. Set the `ReplaceValues` name-value argument to `false` for these functions:

- `smoothdata`
- `normalize`
- `filloutliers`
- `fillmissing`
- `standardizeMissing`

### **Data Preprocessing Functions: Return table with logical values using the `OutputFormat` name-value argument**

When you preprocess tables and timetables, you can now output a table or timetable containing logical values instead of a logical array. Set the `OutputFormat` name-value argument to `"tabular"` for these functions:

- `ischange`
- `islocalmax`
- `islocalmin`

- `ismissing`
- `isoutlier`

## **ismissing, rmissing, and groupsummary Functions: Accept data types with no standard missing value**

The `ismissing` syntax `ismissing(A)` now returns logical 0 (false) when the input data type has no default definition of a standard missing value.

`rmissing` and the `nummissing` and `nnz` methods of `groupsummary` no longer error for input data types with no default definition of a standard missing value.

An example of code that used to error but now executes is:

```
A = [struct struct struct];
TF = ismissing(A)
```

```
TF =
    1x3 logical array
    0     0     0
```

## **Version History**

Some input types that used to throw an error now execute. If your code relies on the errors that MATLAB threw for those inputs, such as within a `try/catch` block, then your code may no longer catch those errors.

## **Functionality being removed or changed**

### **Plot a variable multiple times in a stacked plot**

*Behavior change*

You can now display the same table or timetable variable multiple times when you call the `stackedplot` function. In previous releases, specifying a variable more than once results in an error.

For example, create a timetable from the `outages.csv` file. Then plot the `RestorationTime` variable under each of the other variables that you specify.

```
tbl = readtimetable("outages.csv");
tbl = sortrows(tbl);
stackedplot(tbl, ["Loss", "RestorationTime", "Customers", "RestorationTime"])
```

### **Live Editor tasks for arrays, tables, and timetables do not run automatically if inputs have more than 1 million elements**

*Behavior change*

Many Live Editor tasks for arrays, tables, and timetables do not run automatically if inputs have more than 1 million elements. In previous releases, these tasks run automatically for input arrays, tables, and timetables of any size. If the inputs have a large number of elements, then the code generated by these tasks can take a noticeable amount of time to run (more than a few seconds).

This change in behavior affects these tasks:

- **Join Tables**
- **Retime Timetable**
- **Stack Table Variables**
- **Synchronize Timetables**
- **Unstack Table Variables**

**Live Editor tasks for preprocessing data do not run automatically if inputs have more than 1 million elements**

*Behavior change*

Many Live Editor tasks for preprocessing data do not run automatically if inputs have more than 1 million elements. In previous releases, these tasks run automatically for input data of any size. If the inputs have a large number of elements, then the code generated by these tasks can take a noticeable amount of time to run (more than a few seconds).

This change in behavior affects these tasks:

- **Clean Missing Data**
- **Clean Outlier Data**
- **Compute by Group**
- **Find Change Points**
- **Find Local Extrema**
- **Normalize Data**
- **Smooth Data**
- **Remove Trends**

## Data Import and Export

### Parquet: Read Parquet file data more efficiently using rowfilter to conditionally filter rows

Conditionally filter and read data faster (Predicate Pushdown) from Parquet files when using `parquetread` and `parquetDatastore`. You can create conditions for filtering by using the `rowfilter` function, `matlab.io.RowFilter` object, and `RowFilter` name-value argument. Due to its metadata-accelerated processing, the `rowfilter` workflow is the recommended approach for filtering Parquet data to import.

### Parquet: Determine and define row groups in Parquet file data

A Parquet file can store a range of rows as a distinct row group for increased granularity and targeted analysis. `parquetread` uses the `RowGroups` name-value argument to determine row groups while reading Parquet file data. `parquetwrite` uses the `RowGroupHeights` name-value argument to define row groups while writing Parquet file data.

### Parquet: Convert, import, and export nested data structures

Use `parquetread` to import nested Parquet file data with:

- `LogicalType` as `LIST`.
- `LogicalType` as `NONE` and `PhysicalType` as either `BYTE_ARRAY` or `FIXED_LEN_BYTE_ARRAY`.

The `parquetread` function converts and imports these data structures as cell arrays.

Use `parquetwrite` to export nested cell arrays as `LIST` arrays. Nested data is beneficial to working with irregularly structured data such as jagged arrays.

### writelines Function: Write plain text to a file

Use the `writelines` function to write a string array or a cell array of character vectors as plain text to a file. The `writelines` function is the writing equivalent of the `readlines` function.

### Reading Online Data: Use web options when reading files over HTTP and HTTPS

Read files over HTTP and HTTPS using the `weboptions` function and specifying the `WebOptions` name-value argument with these functions:

- `readtable`
- `readtimetable`
- `readvars`
- `readstruct`
- `readmatrix`
- `readcell`

- readlines

## **Opus Files: Work with Opus (.opus) audio files.**

Use `audioread`, `audiowrite`, and `audioinfo` to read, write, and analyze Ogg Opus audio files.

## **HDF5 Interface: Write datasets using dynamically loaded filters**

You can read and write HDF5 datasets using dynamically loaded filters with both the high-level and low-level interfaces. For details, see “Import HDF5 Files” and “Export to HDF5 Files”.

The `h5create` function introduces two name-value arguments, `CustomFilterID` and `CustomFilterParameters`, to enable compression using dynamically loaded filters.

## **NetCDF Interface: Enable byte-range reading of remote datasets**

You can now use the existing high-level and low-level interfaces for read-only access to remote datasets using the HTTP byte-range capability. The latter assumes that the remote server supports byte-range access.

## **NetCDF Interface: Read and write variable length array data types (NC\_VLEN)**

You can now use the existing high-level functions to read variable length array data types (NC\_VLEN) from NetCDF-4 files. You can read and write NC\_VLEN types using low-level functions.

Use these additional low-level functions to create NC\_VLEN types and retrieve information about them:

- `netcdf.defVlen`
- `netcdf.inqUserType`
- `netcdf.inqVlen`

## **Scientific File Format Libraries: NetCDF library is upgraded**

The NetCDF library is upgraded to version 4.8.1.

## **Hardware Manager App: Discover and connect to your hardware from MATLAB**

The new **Hardware Manager** app allows you to discover and connect to your hardware from MATLAB by providing access to the necessary add-ons and apps. For more information, see “Get Started with Hardware Manager”.

## **TCP/IP Client Interface: Specify transfer delay options**

You can now enable or disable a transfer delay to allow delayed acknowledgement from the connected server for `tcpclient` objects and in the **TCP/IP Explorer** app. The transfer delay is

enabled by default. Enabling the delay turns on Nagle's algorithm, which causes the client to collect small segments of outstanding data and send them in a single packet when acknowledgement (ACK) arrives from the server. Disabling it turns off Nagle's algorithm, which immediately sends data to the network.

For the `tcpclient` interface, you can set the `EnableTransferDelay` property as a name-value argument during object creation. For **TCP/IP Explorer**, you can select **Transfer Delay** options during connection configuration.

For more information about this functionality, see `EnableTransferDelay` and “Configure Connection in TCP/IP Explorer”.

## Functionality being removed or changed

### serialist function will be removed

*Warns*

`serialist` will be removed. Use `serialportlist` instead. For more information about updating your code to use the recommended functionality, see “Transition Your Code to serialport Interface”.

### serial function will be removed

*Warns*

`serial` and its object properties will be removed. Use `serialport` and its properties instead.

This example shows how to connect to a serial port device using the recommended functionality.

Functionality	Use Instead
<pre>s = serial("COM1"); s.BaudRate = 115200; fopen(s)</pre>	<pre>s = serialport("COM1",115200);</pre>

For more information about updating your code to use the recommended functionality, see “Transition Your Code to serialport Interface”.

### MATLAB Variable Editor: timeseries will no longer be supported

*Warns*

Viewing `timeseries` objects using the MATLAB Variable Editor will no longer be supported. To view time-indexed data in the Variable Editor, use `timetable` instead.



## Mathematics

### **pagemldivide, pagemrdivide, and pageinv Functions: Solve linear equations and calculate matrix inverses using pages of N-D arrays**

Use the `pagemldivide`, `pagemrdivide`, and `pageinv` functions to perform linear algebra operations on the pages of N-D arrays. In this context, the N-D array is treated as a container for several 2-D matrices.

- `pagemldivide` and `pagemrdivide`: Solve linear equations using the pages of N-D arrays.
- `pageinv`: Calculate the matrix inverse of the pages of an N-D array.

### **tensorprod Function: Calculate tensor products between two arrays**

Use the `tensorprod` function to calculate tensor products between two N-D arrays. You can perform an inner product, outer product, or a combination of the two by specifying a subset of dimensions to contract (multiply and sum) with each other.

### **round Function: Control tiebreak behavior**

The `round` function has a new `TieBreaker` name-value argument to specify how to break ties. You can now specify to round ties away from zero, towards zero, to the nearest even or odd integer, or towards positive or negative infinity.

### **Version History**

Starting in R2022a, the `round` function always rounds ties (that are within roundoff errors) away from zero by default. In previous releases, the `round` function sometimes returns inconsistent results, where ties are rounded towards zero by default.

### **null and orth Functions: Specify tolerance to treat singular values below a threshold as zero**

The `null` and `orth` functions now have a second input argument that specifies a tolerance. The tolerance determines which singular values of the input matrix are treated as zero, which can change the number of columns returned by `null` and `orth`.

### **norm Function: Frobenius norm calculations support N-D arrays**

Frobenius norm calculations of the form `norm(X, "fro")` now support N-D arrays. See `norm` for more information.

### **equilibrate Function: Specify output format of factorization**

`equilibrate` now has an option with values of `"vector"` or `"matrix"` to specify whether the output arguments are returned as vectors or matrices. For large factorizations, returning the outputs as vectors can save memory and improve efficiency.

## **rand, randi, and randn Functions: Support for complex input and RandStream object with the "like" syntax**

The `rand`, `randi`, and `randn` functions now support complex input and a `RandStream` object for the "like" syntax.

For example, you can use `X = rand(m,n,"like",p)` to create an  $m$ -by- $n$  array of random numbers of the same data type and complexity (real or complex) as `p`. You can also use `X = rand(s,m,n,"like",p)` to generate random numbers like `p` from the random number stream `s` (`RandStream` object) instead of the default global stream.

## **eps, flintmax, intmax, intmin, realmax, and realmin Functions: Use "like" syntax to return scalars based on prototype object**

The `eps`, `flintmax`, `intmax`, `intmin`, `realmax`, and `realmin` functions now accept the "like" syntax to return scalars based on a prototype object.

For example, you can use `f = realmax("like",p)` to return the largest finite floating-point number with the same data type, sparsity, and complexity (real or complex) as the floating-point variable `p`.

## **qr and gsvd Functions: Option for economy-size decompositions**

`qr` and `gsvd` have a new "econ" option for economy-size decompositions.

- For `qr`, the functionality is the same as `qr(A,θ)` unless a third output is specified.
- For `gsvd`, the functionality is the same as `gsvd(A,B,θ)`.

## **Functionality being removed or changed**

### **One-output qr syntax always returns upper-triangular factor**

*Behavior change*

The syntax `R = qr(A)` always returns `R` as an upper-triangular matrix, regardless of whether `A` is full or sparse. Previously, for full `A`, the one-output syntax returned an `R` matrix with intermediate data used in the calculation located in the lower triangular portion of the matrix. See `qr` for more information.

### **mldivide no longer uses LDL factorization for full matrices**

*Behavior change*

`mldivide` no longer uses an LDL factorization for full matrices that are Hermitian indefinite. Instead, the LU factorization is used for these matrices.

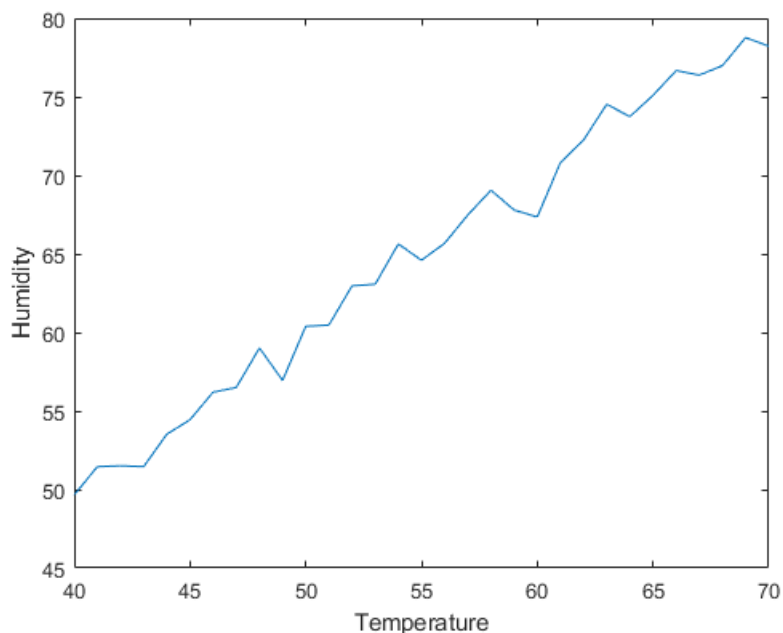
## Graphics

### Plotting Table Data: Create line plots by passing tables directly to plotting functions

Create plots by passing a table directly to any of these functions: `plot`, `plot3`, `loglog`, `semilogx`, `semilogy`, and `polarplot`. When you specify your data as a table, Cartesian axis labels and the legend (if present) are automatically labeled using the table variable names.

For example, create a table with the variables `Temperature` and `Humidity`. Pass the table to the `plot` function as the first argument, and specify the variables you want to plot.

```
Temperature = (40:70)';
Humidity = (50:80)' + randn(31,1);
T = table(Temperature,Humidity);
plot(T,"Humidity","Temperature")
```

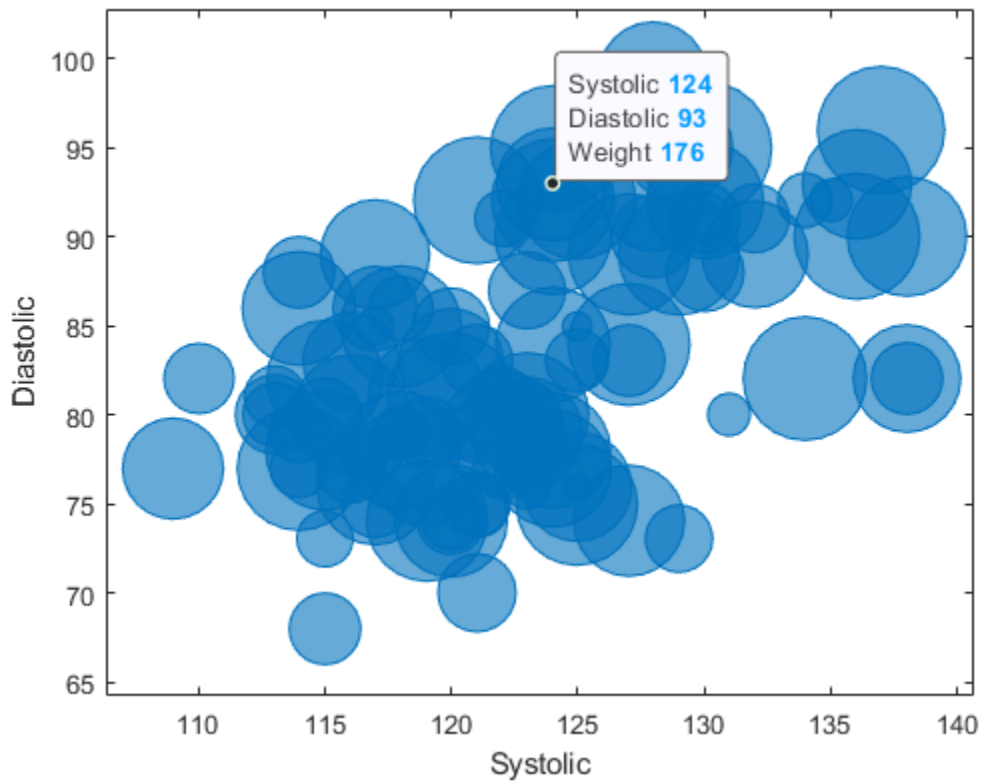


### Data Tips: View table variable names as row labels

When plotting tabular data, the default row labels of data tips created interactively or with the `datatip` function are the names of the table variables associated with the data point.

For example, create a table using the sample file `patients.xls`. Then, plot the `Systolic`, `Diastolic`, and `Weight` variables in a bubble chart. A data tip created with `datatip(b)` displays three rows. The row labels are "Systolic", "Diastolic", and "Weight".

```
tbl = readtable("patients.xls");
b = bubblechart(tbl,"Systolic","Diastolic","Weight");
datatip(b);
```

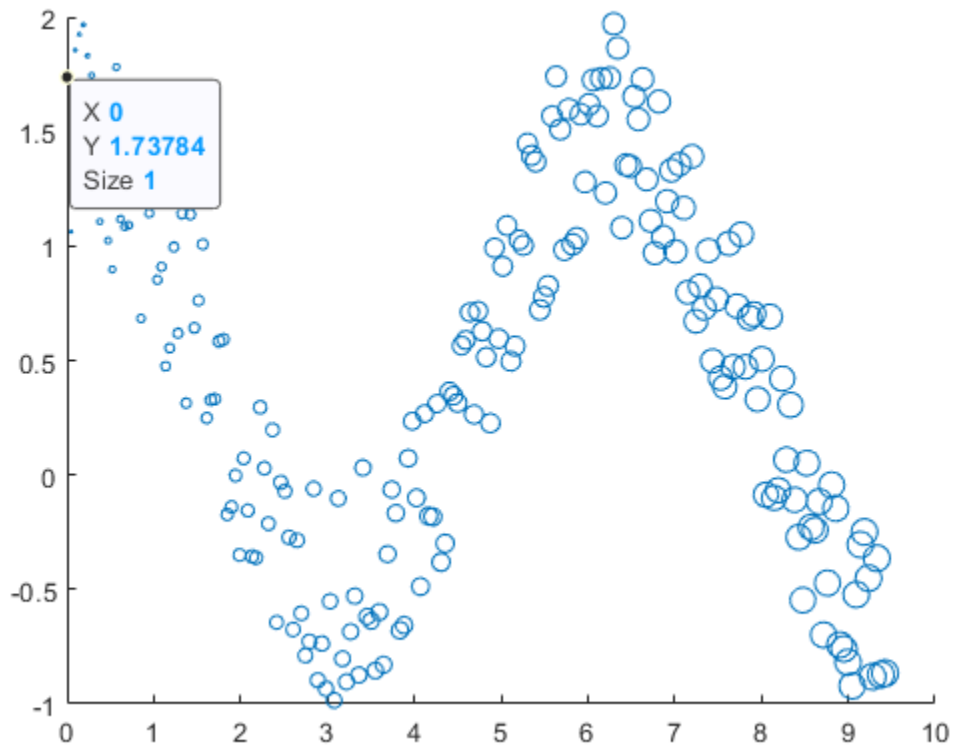


## Data Tips: View visual property values for scatter plots and bubble charts

For scatter plots and bubble charts, data tips created interactively or with the `datatip` function include by default rows for visual properties such as size, color, or transparency that are specified with vector data.

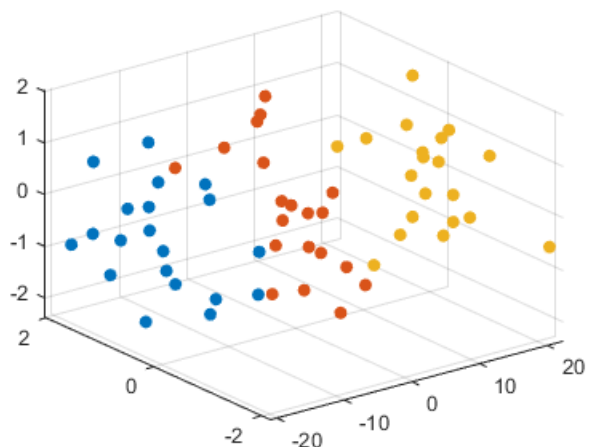
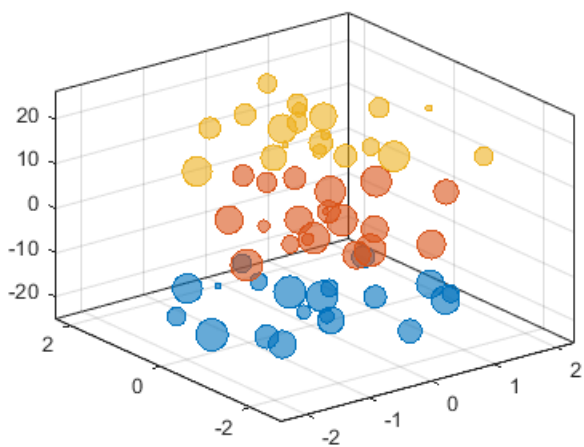
For example, create a scatter plot of random data and define the marker sizes as vector `sz`. A data tip created with `datatip(s)` displays three rows: X, Y, and Size. The Size row in the data tip displays the marker size specified by `sz` for the associated data point.

```
x = linspace(0,3*pi,200);  
y = cos(x) + rand(1,200);  
sz = linspace(1,100,200);  
s = scatter(x,y,sz);  
datatip(s);
```



## Bubble Charts and 3-D Scatter Plots: Plot multiple data sets at once

The `bubblechart`, `bubblechart3`, `polarbubblechart`, and `scatter3` functions now accept the same combinations of matrices and vectors as the `plot` function does. As a result, you can visualize multiple data sets at once rather than using the `hold` function between plotting commands.



## fontname and fontsize Functions: Specify the font and font size for graphics objects

Use the `fontname` and `fontsize` functions to modify the fonts displayed with graphics objects such as figures, axes, legends, tiled chart layouts, standalone visualizations, and UI components. MATLAB applies your changes to the specified object and all the objects it contains. For example, if you change the font on a figure, all the axes, annotations, and UI components within the figure use the new font.

## exportgraphics Function: Create animated GIF files

Create animated GIF files by calling the `exportgraphics` function multiple times with the Append name-value argument.

## Annotation Graphics Objects: Change the annotation rotation angle with the Rotation property

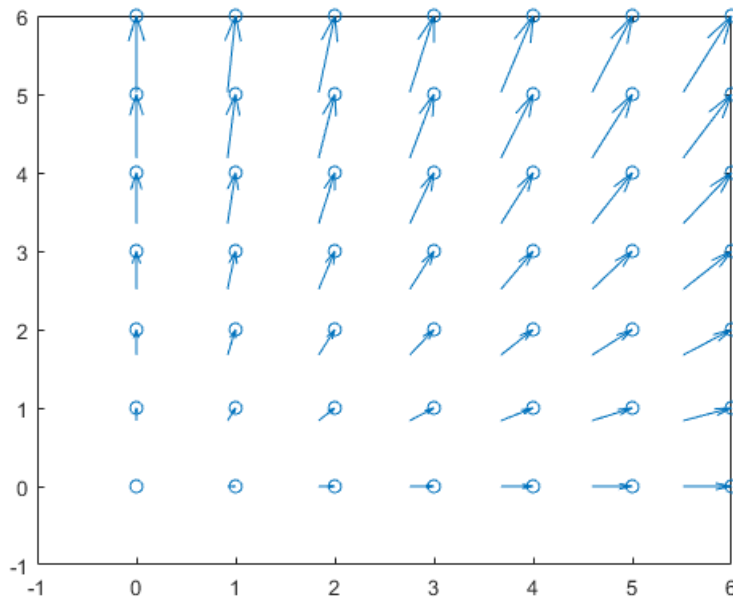
For text box, rectangle, and ellipse annotation objects, rotate the annotation a specified number of degrees by using the `Rotation` property. The anchor point for rotation is the location specified by the first two elements of the `Position` property, so the `Position` property is unaffected by rotation.

For more information, see [TextBox Properties](#), [Rectangle Properties](#), and [Ellipse Properties](#).

## Quiver Plots: Align the heads, centers, or tails of arrows with data points

Set the `Alignment` property of a `Quiver` object to control how the arrows align with the data points. Valid values are "head", "center", and "tail". For example, plot a grid of vectors with the arrow heads positioned at the data points. Specify a marker symbol to show the locations of the data points.

```
[X,Y] = meshgrid(0:6,0:6);  
U = 0.25*X;  
V = 0.5*Y;  
quiver(X,Y,U,V,"Alignment","head","Marker","o")
```



## **xlim, ylim, and zlim Functions: Query the axis limit method**

Query the method MATLAB uses to set the axis limits by calling the `xlim`, `ylim`, and `zlim` functions and specifying "method" as an input argument.

## **view Function: Change the view on multiple axes simultaneously**

Change the view of multiple axes objects at the same time by passing an array of axes objects to the `view` function.

## **rendererinfo Function: Get renderer information without specifying the axes**

Call the `rendererinfo` function without any arguments to query the default graphics renderer information. This new syntax allows you to call the `rendererinfo` function in a way that is consistent with the `opengl` syntax. Since R2019a, the `rendererinfo` function has been recommended instead of the `opengl` function for querying the renderer.

## **linkaxes Function: Synchronize axes in all dimensions by default**

The `linkaxes` function now supports 3-D Cartesian axes and synchronizes the x-axis, y-axis, and z-axis limits by default. The supported values of the `dimension` input argument are now 'xyz' (default), 'x', 'y', 'z', 'xy', 'xz', 'yz', and 'off'.

Before R2022a, `linkaxes` supported only 2-D Cartesian axes and synchronized the x-axis and y-axis limits by default.

## cameratoolbar Function: Syntax support for figures created with the uifigure function

Syntaxes of the `cameratoolbar` function that do not directly make the toolbar visible are now supported by figures created with the `uifigure` function.

## Callbacks in Live Editor: Create callbacks for figures in the Live Editor

You can now create callbacks for figures created in the Live Editor. The callback workflow supports optional source and event-data parameters.

Keyboard-based callback properties and anonymous function callbacks using `Figure` objects from the MATLAB workspace are not currently supported in the Live Editor.

To define and execute a figure callback in the Live Editor, use one of these techniques:

- Create a figure callback and pass source and event data as parameters in the callback.
- Create a figure callback and do not pass source or event data as a parameter in the callback.
- Create a callback that includes a function for identifying a graphics object, such as `gca` or `findobj`.

For example, define a callback function called `colorchangeCallback`. With the `colorchangeCallback` function on the MATLAB path, use the `@` operator to assign the function handle to the `WindowButtonDownFcn` property of the figure `fig`.

```
fig = figure;
axis([-4 4 -4 4]);
plot(1:10)
fig.WindowButtonDownFcn = @(src,eventdata)colorchangeCallback(src);
```

Define the callback and set the `Color` property for the `Axes` object in the figure:

```
function colorchangeCallback(f,~)
% Change the axes color on button down
ax = f.Children;
ax.Color = rand(1,3);
end
```

For more information, see “Callbacks in Live Editor”.

## Figure Code: Generate code for figure interactions in MATLAB Online

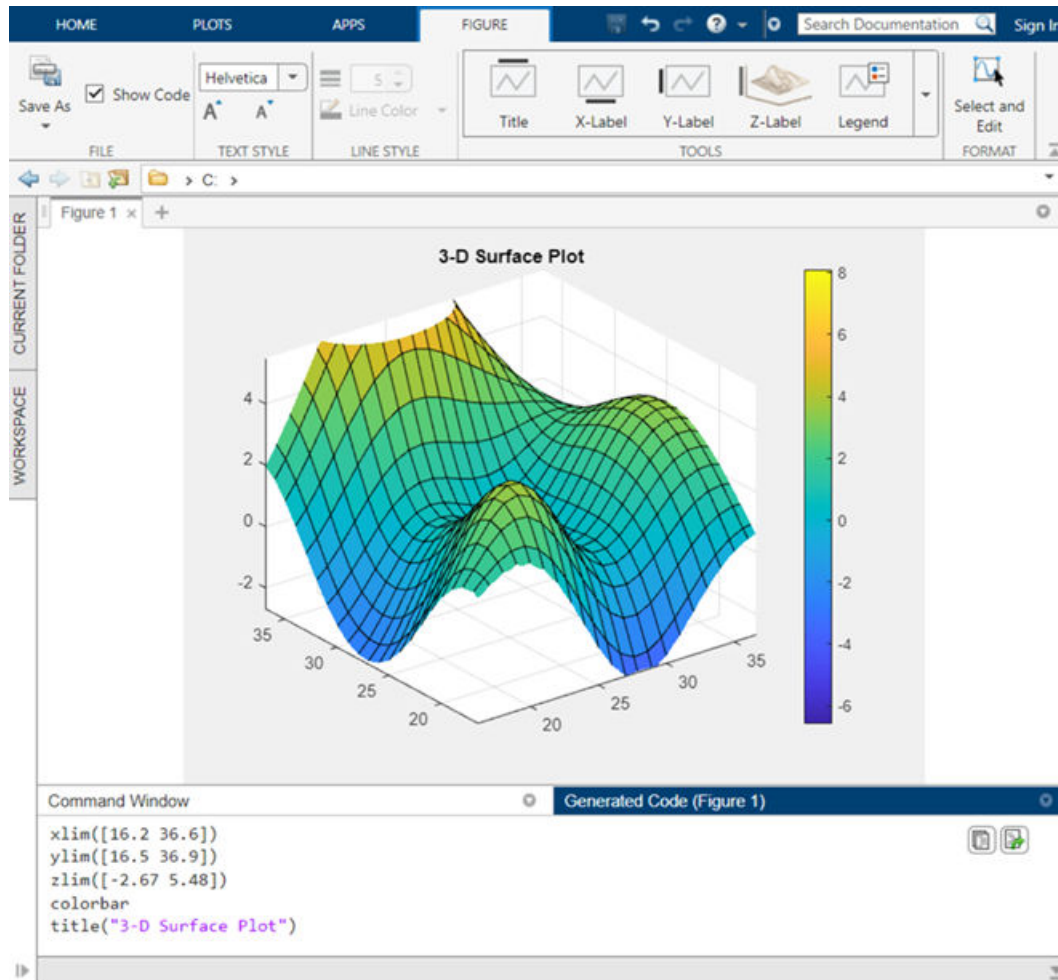
When you modify a figure in MATLAB Online using the **Figure** tab, MATLAB generates code that you can view, copy, and export. To view the generated code, select **Show Code** in the **File** section of the **Figure** tab. MATLAB generates code for these actions:

- Adding a title, axis label, legend, color bar, grid, or annotation
- Changing the text or line style
- Using the pan, zoom, rotate, or data tip interactions

MATLAB does not currently generate code for the **Select and Edit** option in the **Figure** tab.



For example, you can create a surface plot with `surf` (peaks). Then, interactively add a title and colorbar, zoom into the axes, and view generated code.



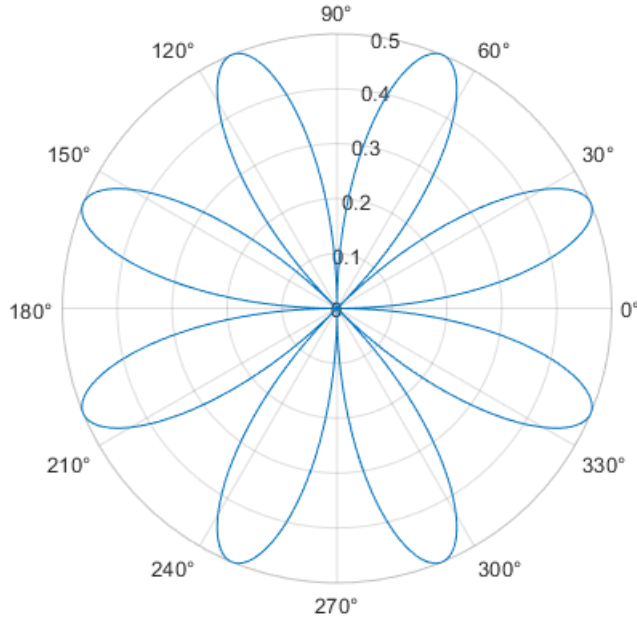
## Functionality being removed or changed

### Polar axes display angle values with degree symbols

#### Behavior change

Polar axes now display tick values in degrees with degree symbols when the `ThetaAxisUnits` property is set to "degrees". For example, create a polar plot. By default, the *theta*-axis displays the tick values with degree symbols.

```
theta = 0:0.01:2*pi;
rho = sin(2*theta).*cos(2*theta);
polarplot(theta,rho)
```



This change clarifies which units are being used for the *theta* tick values. You can use the `ThetaAxisUnits` property to display the tick values in degrees or radians. To remove the degree symbols, change the tick label format for the *theta*-axis:

```
pax = gca;
pax.ThetaAxis.TickLabelFormat = "%g";
```

### The `caxis` function is not recommended

*Still runs*

The `caxis` function is no longer recommended. However, the function continues to work, and there are no plans to remove it at this time.

To update your code, call the `clim` function instead. It accepts the same input arguments and returns the same output as the `caxis` function.

### The `im2java` function will be removed

*Still runs*

`im2java` will be removed in a future release. There is no replacement for this function.

### The Plot Catalog tool will be removed

*Still runs*

The Plot Catalog tool will be removed in a future release. Instead, to interactively create and explore visualizations for your data, use the **Plots** tab in the MATLAB Toolstrip or the **Create Plot** task in the Live Editor.

For more information about visualizations, see “Types of MATLAB Plots” or toolbox-specific documentation.

### The `opengl` function will be removed

*Still runs*

The `opengl` function will be removed in a future release.

- To query the renderer, use the `renderinfo` function instead of the `opengl` function.
- Changing the renderer with the `opengl` function will no longer be necessary when the function is removed.

### **The renderer startup options will be removed**

*Still runs*

In a future release, the MATLAB startup options for selecting the graphics renderer will be removed. Specifically, these startup scenarios will no longer be available:

- `matlab -softwareopengl`
- `matlab -nosoftwareopengl`
- `matlab -softwareopenglmesa`
- `matlab -noopengl`

It will no longer be necessary to specify the renderer when these options are removed.

### **The Renderer property of figures will have no effect**

*Behavior change in future release*

The `Renderer` and `RendererMode` properties of figures will have no effect in a future release. It will no longer be necessary to change the renderer when these properties are disabled.

### **The FontSmoothing property will have no effect**

*Behavior change in future release*

The `FontSmoothing` property for all types of axes, rulers, geographic scales, and text objects will have no effect in a future release. Font smoothing will be enabled regardless of the value of the property.

### **Some plot tools functions will redirect to the Figure Toolstrip and Property Inspector**

*Behavior change in future release*

Calling these plot tools functions will open a configuration of the Figure Toolstrip and Property Inspector. For more information, see the Version History section in the documentation for each function.

- `figurepalette`
- `plotbrowser`
- `propertyeditor`
- `propedit`
- `plottools`
- `showplottool`
- `plotedit`

Currently, calling plot tools functions opens the Figure Palette, Plot Browser, and Property Editor.

## App Building

### uistyle Function: Add icons and format text in table cells and tree nodes

You can now create styles for table and tree UI components that specify an icon and a text interpreter using the `uistyle` function.

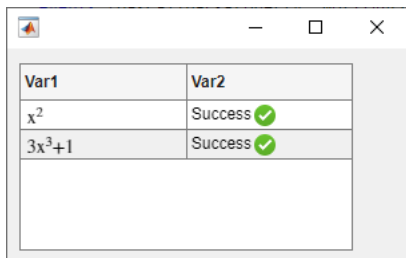
- Specify the `Icon` property of the style object to add icons to table cells and tree nodes.
- Specify the `IconAlignment` property of the style object to modify where the icon appears in relation to the text in table cells.
- Specify the `Interpreter` property of the style object to format text or add links using HTML markup, or to add equations using TeX or LaTeX markup to table cells and tree nodes.
- Specify the `HorizontalClipping` property of the style object to control whether long text is clipped on the left or the right in table cells and tree nodes.



Add a style to a UI component using the `addStyle` function.

For example, this code creates two styles, one that specifies an icon and one that specifies the text interpreter as TeX, and applies the styles to columns of a table.

```
T = table(["x^2";"3x^3+1"],["Success";"Success"]);
fig = uifigure(Position=[500 500 300 160]);
t = uitable(fig,Position=[10 10 250 140],Data=T);

s1 = uistyle(Interpreter="tex");
s2 = uistyle(Icon="success",IconAlignment="right");
addStyle(t,s1,column=1)
addStyle(t,s2,column=2)
```



Var1	Var2
$x^2$	Success 
$3x^3+1$	Success 

### uitable Function: Rearrange columns of table UI components interactively

You can specify the ability to interactively rearrange table columns in an app by using the `ColumnRearrangeable` property. In a table UI component with the `ColumnRearrangeable` value set to 'on', rearrange table columns in the app by clicking and dragging the column header.

In App Designer and apps created using the `uifigure` function, you can program an app to respond when a user rearranges table columns by creating a `DisplayDataChangedFcn` callback function.

For more information, see [Table Properties](#).

## focus Function: Give keyboard focus to UI components programmatically

Use the `focus` function to programmatically give focus to keyboard-focusable UI components. When a UI component is focused, it is displayed with a blue focus ring, and app users can interact with the component using the keyboard.

## isInScrollView Function: Determine if a component is visible in a scrollable container

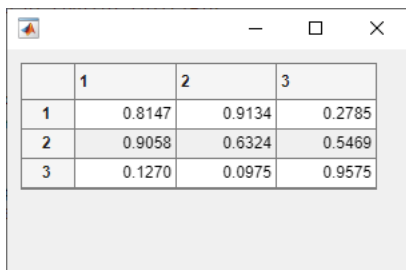
Use the `isInScrollView` function to programmatically identify which components are visible given the size and scroll location of a scrollable container. For example, you can determine which axes are visible inside a scrollable figure window and then update the data only for those axes.

## uigridlayout Function: Resize table, list box, and image UI components to fit content

Grid layout managers with row heights or column widths of `'fit'` now resize to fit the contents of table, list box, and image UI components.

For example, when you create a table UI component inside a grid layout manager with a row height or column width of `'fit'`, the height of the row or the width of the column resizes to fit the data in the table.

```
fig = uifigure(Position=[680 558 300 170]);
gl = uigridlayout(fig);
gl.RowHeight = {'fit'};
gl.ColumnWidth = {'fit'};
tbl = uitable(gl,Data=rand(3));
```



	1	2	3
1	0.8147	0.9134	0.2785
2	0.9058	0.6324	0.5469
3	0.1270	0.0975	0.9575

## Version History

In R2021b, grid layout managers with row heights or column widths of `'fit'` scaled to a fixed size when the row or column contained a table, list box, or image UI component.

- Table UI component — Row height and column width previously resized to 300 pixels.
- List box UI component — Row height previously resized to display at most four items. The exact pixel value to display four items might vary depending on your settings.
- Image UI component — Row height and column width previously resized to 100 pixels.

To display a table, list box, or image at its size in a release before R2022a, set the corresponding elements of the `RowHeight` and `ColumnWidth` properties of the `GridLayout` object to their respective fixed sizes.

## Live Editor Tasks: Develop your own Live Editor tasks for use in live scripts and functions

Live Editor tasks are simple point-and-click interfaces that can be embedded into a live script. Tasks represent a series of MATLAB commands that are automatically generated as users explore parameters.

You can develop your own custom Live Editor tasks by creating a subclass of the `LiveTask` base class. Develop tasks to perform your own specific set of operations within a live script.

For more information, see “Live Editor Task Development Overview”.

## Custom UI Components: Interactively create custom UI components in App Designer

Use App Designer to interactively build your own UI components. Open a new blank custom UI component in App Designer, lay out the component by combining existing MATLAB UI components or graphics objects, and configure the component interface by creating public properties and public callbacks that can be set when the component is used in an app.

Creating a custom UI component has these benefits:

- **Modularization** — Separate the display of large apps into independent, maintainable pieces.
- **Reusability** — Provide a convenient interface for adding and customizing similar components in apps.
- **Flexibility** — Extend the appearance and behavior of existing UI components.


For more information, see “Create a Simple Custom UI Component in App Designer”.

## App Designer: Modify tab focus order of components

You can view and modify the order in which components in your app receive keyboard focus when the app user presses **Tab**. First, sort and filter the **Component Browser** by tab order by selecting **Sort & Filter by Tab Order** from the drop-down list labeled **View**. The **Component Browser** lists only the components in the app that can have focus, in the order of focus. You can then change the tab order of the components by clicking and dragging the component names in the **Component Browser**.

Alternatively, App Designer can automatically apply a left-to-right and then top-to-bottom tab focus order for components in a container. Right-click the name of the container in the **Component Browser** and select **Apply Auto Tab Order**.

## App Designer: Specify error handling options and navigate from error messages when debugging an app

To specify error handling options when debugging code in App Designer, configure the  **Run** button by clicking **Run**. You can choose to pause code execution when an error occurs, when a warning occurs, or when a NaN or Inf value is returned.

Additionally, error messages in App Designer now contain links to relevant files and functions. Use these links to navigate more easily to the documentation or to line numbers in your code when debugging your apps.

## App Designer: Manage image files in your app with an improved workflow

When you specify image data for your app, such as the image source of an image component or the icon of a button, select an image that is in the same folder as the MLAPP file or one of its subfolders. The image will then load whenever the app is opened or run without it needing to be on the MATLAB path. Alternatively, you can continue to use images in any location by adding the image files to the MATLAB path.

## App Designer: Convert components in a grid layout manager to use pixel-based positioning

You can delete a grid layout manager and convert the components in the grid to use pixel-based positioning. To use pixel-based positioning when you were previously using a grid layout manager, right-click the container with the grid layout manager in the canvas, and select **Remove Grid Layout**.

For more information, see “Use Grid Layout Managers in App Designer”.

## App Designer: Use App Designer in most modern web browsers in MATLAB Online

You can now use App Designer in MATLAB Online with the current versions of Mozilla® Firefox®, Apple Safari, and Microsoft Edge®, in addition to Google Chrome™.


## App Designer: Customize design environment layout

You can now customize the locations of the side panels and tabs in the App Designer design environment.

To change the location of side panels such as the **Component Library** and the **Component Browser**, click the panel header and drag it to a new location in the App Designer environment. To change the location of your open tabs to display on the left, right, or bottom of the working area, right-click the tab bar and select **Tab Position**.

Your changes to the design environment layout now persist even after you close and reopen App Designer.

## Comparison Tool: Compare and merge app files in MATLAB Online

Compare and merge two versions of an app file in MATLAB Online using the Comparison Tool. To open the Comparison Tool, click  **Compare** in the **Designer** tab of the App Designer Toolstrip. For more information, see “Compare and Merge Apps”.

## Functionality being removed or changed

### RearrangeableColumns property of table UI components is not recommended

*Still runs*

Starting in R2022a, using the `RearrangeableColumns` property to specify the ability to rearrange columns in a table UI component is not recommended. Use the `ColumnRearrangeable` property instead. The new property can have the same values as the old one.

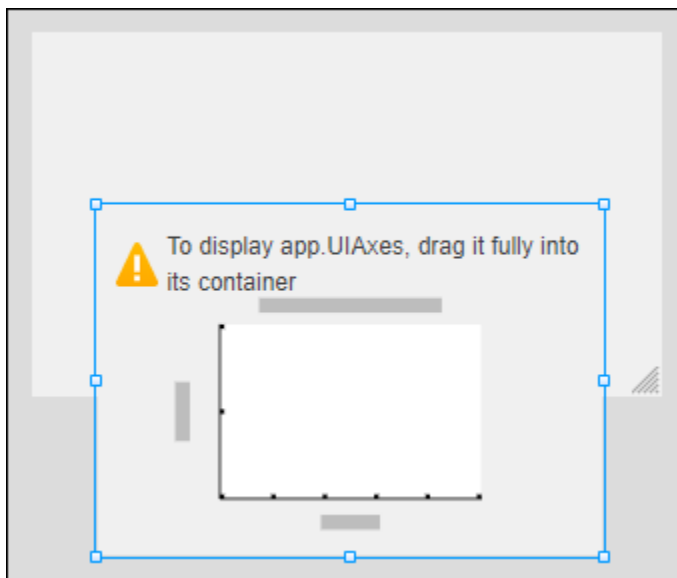
There are no plans to remove support for the `RearrangeableColumns` property at this time. However, the `RearrangeableColumns` property no longer appears in the list returned by calling the `get` function on a table UI component.

### UIAxes content is not displayed in App Designer Design View when the component is off the canvas

*Behavior change*

When creating an app with a `UIAxes` component in App Designer, the `UIAxes` component content is not displayed in **Design View** when the component is partially or fully off the canvas. Instead, the `UIAxes` component is shown as a placeholder image. To see the content of the component, drag it fully onto the canvas.

You can still modify properties of the `UIAxes` component when it is off the canvas, but you will not be able to see a visual reflection of those changes in **Design View** until the component is dragged onto the canvas.





## ComponentContainer class assigns a parent before executing the setup method

### *Behavior change*

When you create an instance of a custom UI component created using the `matlab.ui.componentcontainer.ComponentContainer` class, the class now assigns the component parent before executing the `setup` method. As a result, you might see unexpected behavior if your `setup` method creates underlying UI components that can be parented to either a figure created using the `figure` function or a figure created the `uifigure` function, such as panels, tab groups, or button groups.

To update your class code, when you create such a component, specify the property value explicitly for any property where the default value differs depending on the parent. For example, to create a panel in your custom component that is sized using normalized units, specify `Units` as "normalized" before setting the `Position` property.

## ButtonDownFcn callback cannot be interactively assigned for custom UI components in App Designer

### *Behavior change*

Starting in R2022a, when you use a custom UI component in an app in App Designer, you cannot interactively assign a `ButtonDownFcn` callback to the component. If you have an existing App Designer app that contains a custom UI component with a `ButtonDownFcn` callback that you assigned interactively, opening the app in R2022a disconnects the callback from the component. To reassign the callback to the component, follow these steps:

- 1 If your app does not contain a `StartupFcn` callback, right-click the app node from the top of the **Component Browser** hierarchy and select **Callbacks > Add StartupFcn Callback**.
- 2 In **Code View**, in the `startupFcn` function, add this code to assign the appropriate `ButtonDownFcn` callback programmatically:

```
app.CustomUIComponentName.ButtonDownFcn = ...
    @(src,event)CallbackFunctionName(app,event);
```

For example, if your app contains a custom UI component named `app.IPAddress` with a `ButtonDownFcn` callback named `IPAddressButtonDown`, add this code to the `startupFcn` function of your app:

```
app.IPAddress.ButtonDownFcn = @(src,event)IPAddressButtonDown(app,event);
```

## Performance

### table Data Type Indexing: Improved performance when subscripting with dot notation or multiple levels of indexing

table subscripting when using dot notation is significantly faster in R2022a than in R2021b. Also, subscripting with multiple levels of indexing is faster.

- For example, when you use dot notation to refer to a table variable with  $10^6$  elements, performance in R2022a is more than 4x faster than in R2021b.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
    indices = 1:1e5;

    tic;
    % Refer to variable using dot notation
    for i = indices
        x = t.Var1;
    end
    toc
end
```

The approximate execution times are:

**R2021b:** 1.55 s

**R2022a:** 0.36 s

- Similarly, when you use dot notation to assign an array to a table variable with  $10^6$  elements, performance in R2022a is about 3x faster than in R2021b.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
    indices = 1:1e5;
    x = randi(1e6,1e6,1);

    tic;
    % Assign to variable using dot notation
    for i = indices
        t.Var1 = x;
    end
    toc
end
```

The approximate execution times are:

**R2021b:** 2.15 s

**R2022a:** 0.72 s

- Also, when you use dot notation and parentheses to assign individual values to elements of a table variable, performance in R2022a is more than 4x faster than in R2021b.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
    indices = randi(1e6,1,1e5);
```

```

tic;
% Assign to elements using dot notation and parentheses
for i = indices
    t.Var1(i) = rand;
end
toc
end

```

The approximate execution times are:

**R2021b:** 5.08 s

**R2022a:** 1.20 s

The code was timed on a Windows 10, Intel® Xeon CPU W-2133 @ 3.60 GHz test system by calling each version of the `timingTest` function.

## Classes: Improved performance for static methods, constant property access, and package functions in scripts

The overhead times for these three actions when performed in scripts are reduced:

- Executing a package function
- Executing a static method
- Accessing a constant property

The overhead times for these actions in a script are now comparable with the times of execution in a function, and the overheads are small enough that they can generally be ignored for performance considerations.

This code performs each of these actions 1,000,000 times. (The code called in each loop is shown at the end of this note.)

```

tic;
for j = 1:1000000
    out = pkg1.pkg2.packageFunction(2);
end
toc;

tic;
for j = 1:1000000
    out = MyClass.staticMethod(1);
end
toc;

tic;
for j = 1:1000000
    out = pkg1.PackageClass.constantProperty;
end
toc;

```

The approximate times to complete each loop are:

**R2021b:**

- Package function: 8.4 s
- Static method: 7.8 s
- Constant property access: 32 s

**R2022a:**

- Package function: 0.04 s (210x faster)
- Static method: 0.031 s (252x faster)
- Constant property access: 0.039 s (821x faster)

The code was timed on a Windows 10, Intel Xeon® CPU W-2133 @ 3.60 GHz test system.

For use with the test code, PackageClass must be in folder +pkg1, and packageFunction must be in folder +pkg1/+pkg2.

```
function out = packageFunction(in)
    out = in;
end

classdef MyClass
    methods (Static)
        function out = staticMethod(in)
            out = in;
        end
    end
end

classdef PackageClass
    properties (Constant)
        constantProperty = 3;
    end
end
```

## try Block: Improved performance when statements run error-free

The try block shows improved performance when the statements within the block run error-free. For example, this code is approximately 6x faster than in the previous release:

```
function testTryPerformance
x = 1;
for i = 1:1e8
    try
        x = x * i;
    catch
        warning("Assignment was not successful.")
        x = 1;
    end
end
end
```

The approximate execution times are:

**R2021b:** 2.3 s

**R2022a:** 0.4 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system using the `timeit` function:

```
timeit(@testTryPerformance)
```

## Python Data Type Conversion: Improved performance when converting between Python and MATLAB data types in out-of-process mode

When you run Python code out of process, conversions between Python data types and MATLAB data types show improved performance.

- The `timingPythonList` function converts a Python `list` object to a MATLAB cell array. This code is about 118x faster than in the previous release:

```
function timingPythonList
    l = py.list(['MATLAB', 'R', 2022, 'a', 'and', 1.2, 10^2, ...
                2021, 12, 25, '2021', '12', '25', 'and', ...
                2021, 12, 25, '2021', '12', '25']);
    tic
    ml = cell(l);
    toc
end
```

The approximate execution times are:

**R2021b:** 1.18 s

**R2022a:** 0.01 s

- The `timingPythonDict` function converts a Python dictionary object to a MATLAB structure. This code is about 57.9x faster than in the previous release:

```
function timingPythonDict
    d = py.dict(pyargs('a', 1, 'b', 2, 'c', 3, 'd', 4, 'e', 5, ...
                      'f', 6, 'g', 7, 'h', 8, 'i', 9, 'j', 10, 'k', 11, 'l', 12, ...
                      'm', 13, 'n', 14, 'o', 15, 'p', 16, 'q', 17, 'r', 18, ...
                      's', 19, 't', 20));
    tic
    ms = struct(d);
    toc
end
```

The approximate execution times are:

**R2021b:** 0.521 s

**R2022a:** 0.009 s

- The `timingDataTransfer` function converts an array with  $10^8$  elements from a MATLAB double array to a Python `memoryview` object and then back to a MATLAB double array. This code is about 10x faster when converting from MATLAB to Python and approximately 11x faster when converting from Python to MATLAB than in the previous release:

```
function timingDataTransfer
    data = rand(100, 10^6);
    tic
```

```

        pydata = py.memoryview(data);
        toc
        tic
        mdata = double(pydata);
        toc
    end

```

The approximate execution times for converting the MATLAB double array to the Python memoryview object are:

**R2021b:** 4.0 s

**R2022a:** 0.4 s

The approximate execution times for converting the Python memoryview object to the MATLAB double array are:

**R2021b:** 7.7 s

**R2022a:** 0.7 s

All of the code was timed on a Windows 10, Intel Core™ i7-10510U CPU @ 1.80 GHz 2.30 GHz test system by using Python 3.9 in out-of-process mode and calling the `timingPythonList`, `timingPythonDict`, and `timingDataTransfer` functions.

## MATLAB Engine API for Python: Improved performance with large multidimensional arrays in Python

The Python multidimensional array component used by the MATLAB Engine API for Python shows improved performance when:

- 1 Converting data from Python sequences to the data types defined by the `matlab` module
- 2 Transferring data back and forth between Python and MATLAB

In both cases, the improvement is noticeable when operating on arrays with at least 10 elements. When transferring data back and forth, the improvement increases as the size of the array increases.

For example, this Python code measures the execution times of two operations:

- 1 Converting a Python array of size  $10^8$  to a MATLAB double array
- 2 Summing the elements of the MATLAB array using the MATLAB engine

The first operation is about 12x faster than in the previous release, and the second operation is about 110x faster than in the previous release:

```

import random
import time
import matlab.engine
eng = matlab.engine.start_matlab()

rand_array = [random.random() for i in range(10**8)]
s0 = time.perf_counter()
array_md = matlab.double(rand_array,size=(1, 10**8))
s1 = time.perf_counter() - s0
print('conversion to matlab.double(): {} seconds'.format(s1))

```

```
s0 = time.perf_counter()
sum_of_elems = eng.sum(array_md,1)
s1 = time.perf_counter() - s0
print('sum(): {} seconds'.format(s1))
```

The approximate execution times for the first operation are:

**R2021b:** 42 s

**R2022a:** 3.6 s

The approximate execution times for the second operation are:

**R2021b:** 210 s

**R2022a:** 1.9 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by using Python `perf_counter()` statements.

## Matrix multiplication: Improved performance when multiplying sparse and full matrices

Matrix multiplication shows improved performance when:

- One of the operands is a sparse matrix, and the other is a full matrix.
- The sparse operand has at least 50,000 nonzero elements.
- The full operand has at least 32 columns (or at least 32 rows when transposed).

The performance improvement arises from added support for multithreading in the operation, and therefore the speedup improves as the matrix size and number of nonzero elements increase.

For example, multiplying a 102,400-by-102,400 sparse matrix with a 102,400-by-128 full matrix on a machine with 6 physical cores is about 2.7x faster than in the previous release.

```
function timingSparseDenseMult
A = delsq(numgrid('S',322));
B = rand(size(A,2),128);
tic
for k = 1:10
    C = A*B;
end
toc
end
```

The approximate execution times are:

**R2021b:** 0.8 s

**R2022a:** 0.3 s

The code was timed on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system by calling the `timingSparseDenseMult` function.

## inv Function: Improved performance when inverting large triangular matrices

The `inv` function shows improved performance when operating on large triangular matrices.

For example, inverting a 5,000-by-5,000 upper triangular matrix is about 3.7x faster than in the previous release.

```
function timingInv
rng default
A = randn(5e3);
[~,R] = lu(A);

tic
Y = inv(R);
toc
end
```

The approximate execution times are:

**R2021b:** 1.1 s

**R2022a:** 0.3 s

The code was timed on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system by calling the `timingInv` function.

## sprand and sprandn Functions: Improved performance when generating random sparse matrices

The `sprand` and `sprandn` functions show improved performance when generating random sparse matrices if the number of nonzero elements in the output is larger than the number of rows.

For example, generating a 10,000-by-10,000 matrix with 10% density of nonzero elements is about 2.5x faster than in the previous release.

```
function timingSprand
n = 1e4;
d = 0.1;
rng default

tic
sprand(n,n,d);
toc
end
```

The approximate execution times are:

**R2021b:** 2.7 s

**R2022a:** 1.1 s

The code was timed on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system by calling the `timingSprand` function.



## fzero Function: Improved performance

The `fzero` function shows improved performance for objective functions specified as function handles. This improvement is most noticeable for functions that take little time to evaluate. The following example, which solves for  $1e5$  roots of a simple function, takes less than one third of the time of the previous release:

```
N = 1e5;
rng default
levels = 1.5*rand(N,1);
out = zeros(N,1);
tic
for i = 1:N
    out(i) = fzero(@(x)myfun(x,levels(i)),[0 2]);
end
toc

function u = myfun(x,lv)
u = x*sin(x) - lv;
end
```

The approximate execution times are:

**R2021b:** 5.83 s

**R2022a:** 1.66 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by running the above script.

## diff Function: Improved performance with large number of elements

The `diff` function shows improved performance when operating on vectors with at least  $10^5$  elements or when operating along the first or second dimension of matrices and multidimensional arrays with at least  $5 \times 10^5$  elements.

For example, this code creates a `double` array with  $2.5 \times 10^7$  elements and calculates differences between adjacent elements. It is approximately 2.4x faster than in the previous release.

```
function timingDiff
rng default
N = 5000;
A = rand(N);

tic
for k = 1:40
    D = diff(A);
end
toc
end
```

The approximate execution times are:

**R2021b:** 2.43 s

**R2022a:** 1.00 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingDiff` function.

## groupsummary, groupfilter, and grouptransform Functions: Improved performance with small group size

Grouping functions `groupsummary`, `groupfilter`, and `grouptransform` show improved performance, especially when the data count in each group is small.

For example, this code performs group summary computations on a matrix with 500 groups with a count of 10 each. It is about 2.18x faster than in the previous release.

```
function timingGroupsummary
data = (1:5000)';
groups = repelem(1:length(data)/10,10)';
p = randperm(length(data));
data = data(p);
groups = groups(p);

tic
for k = 1:300
    G = groupsummary(data,groups,"mean");
end
toc
end
```

The approximate execution times are:

**R2021b:** 2.14 s

**R2022a:** 0.98 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingGroupsummary` function.

## nufftn Function: Improved performance with nonuniform sample points or query points

The `nufftn` function shows improved performance when operating on either nonuniformly spaced sample points or nonuniformly spaced query points.

For example, this code constructs a 32,768-by-3 matrix of nonuniform sample points `t` and calculates the nonuniform discrete Fourier transform along each dimension of a 32-by-32-by-32 array. The code is about 14.5x faster than in the previous release.

```
function timingSamplePoints
rng default
t = rand(32^3,3);
X = rand(32,32,32);
tic
    Y = nufftn(X,t);
toc
end
```

The approximate execution times are:

**R2021b:** 2.76 s

**R2022a:** 0.19 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingSamplePoints` function.

As another example, this code constructs a 65,536-by-3 matrix of nonuniform query points `f` and calculates the nonuniform discrete Fourier transform along each dimension of a 64-by-32-by-32 array. The code is about 42.6x faster than in the previous release.

```
function timingQueryPoints
rng default
f = rand(64*32*32,3);
X = rand(64,32,32);
tic
    Y = nufftn(X,[],f);
toc
end
```

The approximate execution times are:

**R2021b:** 4.26 s

**R2022a:** 0.10 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingQueryPoints` function.

## Variables Editor and Live Editor: Improved speed of data display when scrolling

For text and datetime data types in the Variables editor or in the generated output of the Live Editor, the performance of vertical and horizontal scrolling is improved. Displayed data is optimized and the rendering mechanism is faster, so data appears more quickly after scrolling in R2022a than in previous releases.

For example, on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system, when you scroll through a timetable with 3000 rows, the displayed data renders more quickly in R2022a than in R2021b.

## App Building: Improved performance when creating UI components

Creating UI components is faster in R2022a than in R2021b. As a result, apps start up faster when you run them. This improvement is more noticeable for apps with many UI components.

For example, this code measures the time it takes to create 500 edit field components. The code is about 1.25x faster than in the previous release.

```
function timingApp
fig = uifigure;
gl = uigridlayout(fig,Scrollable="on");
gl.RowHeight = repmat({'fit'},1,50);
gl.ColumnWidth = repmat({'fit'},1,10);
```

```
drawnow

tic
for k = 1:500
    uieditfield(gl);
end
drawnow
toc
end
```

The approximate execution times are:

**R2021b:** 7.5 s

**R2022a:** 6.0 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingApp` function.

## uitable Function and UI Containers: Improved performance when updating properties successively

Updating property values for certain UI components and containers is faster in R2022a than in R2021b. This performance improvement applies to table UI components created using the `uitable` function, UI containers created using the `uipanel`, `uibuttongroup`, `uitab`, and `uitabgroup` functions, and custom UI components created using the `ComponentContainer` base class, when these objects are parented to a figure created using the `uifigure` function.

For example, this code creates a table UI component and then updates the table cell values for 1000 cells. The code is about 5.2x faster than in the previous release.

```
function timingTableUpdates
fig = uifigure;
tbl = uitable(fig, "Data", rand(1000,15));
drawnow

tic
for k = 1:1000
    tbl.Data(k,1) = 0;
end
drawnow
toc
end
```

The approximate execution times are:

**R2021b:** 2.6 s

**R2022a:** 0.5 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingTableUpdates` function.

## UI Components: Improved performance when setting a property with an unchanged value

Setting a property value of a UI component when the value is unchanged is faster in R2022a than in R2021b. This speed increase leads to improved performance in apps that update many UI component properties at once, even if not all property values have changed.

For example, this code sets the `Value` property of an edit field to the same value 1000 times. The code is about 23x faster than in the previous release.

```
function timingValueSet
fig = uifigure;
ef = uieditfield(fig);
drawnow

tic
for k = 1:1000
    ef.Value = "Text";
    drawnow
end
toc
end
```

The approximate execution times are:

**R2021b:** 2.3 s

**R2022a:** 0.1 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timingValueSet` function.

## App Designer: Improved performance when loading apps with UIAxes components off the canvas

In App Designer, apps with `UIAxes` components that lie partially or fully off the canvas in **Design View** have these performance improvements:

- The app loads faster when you open it.
- The `UIAxes` components update faster in response to property edits.

The speed increase improves as the number of `UIAxes` components that lie off the canvas increases.

For example, create an app in App Designer using these steps:

- 1 Drag five `UIAxes` components onto the canvas.
- 2 Drag each of the axes components so that they lie partially off the canvas.
- 3 Save and close the app.

Reopen the app. The time it takes for App Designer to fully load the app is about 5.2x faster than in the previous release.

The approximate load times are:

**R2021b:** 10.5 s

**R2022a:** 2 s

Once the app is fully loaded, select a UIAxes component and change its title in the Property Inspector. The property updates almost instantaneously. In the previous release, the property update takes approximately 2 seconds.

These interactions were timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system.

## Plots in Apps: Improved responsiveness for event-driven updates in apps

Apps that involve continuous event-driven updates, such as animations implemented with a timer, are more responsive to those events. To observe the improvement, the animation must be created in an app or in a figure created with the `uifigure` function. For example, an app containing an animation controlled by a timer object is more responsive when you call the timer's `start` and `stop` methods.

This code creates an app using a timer object to plot a random number every 0.01 second. If you run this app on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system, the animation responds more quickly when you click the **Start/Stop** button than in the previous release.

```
function mytimerapp
uif = uifigure("CloseRequestFcn",@CloseRequest);
uibutton(uif,"State","Position",[235 59 100 22], ...
    "ValueChangedFcn",@StateButtonChanged,"Text","Start/Stop");
ax = uiaxes(uif,"Position",[25 100 508 300],"XDir","reverse");

% Create initial plot line
p = plot(ax,0:60,zeros(1,61));

% Create timer object
RandTimer = timer(...
    "ExecutionMode","fixedRate", ...
    "Period",0.01, ...
    "BusyMode","queue", ...
    "TimerFcn",@RandTimerFcn);

% Timer function
function RandTimerFcn(~,~,~)
    % Generate a random number and update plot line
    ydata = p.YData;
    ydata = circshift(ydata,1);
    ydata(1) = rand;
    p.YData = ydata;
end

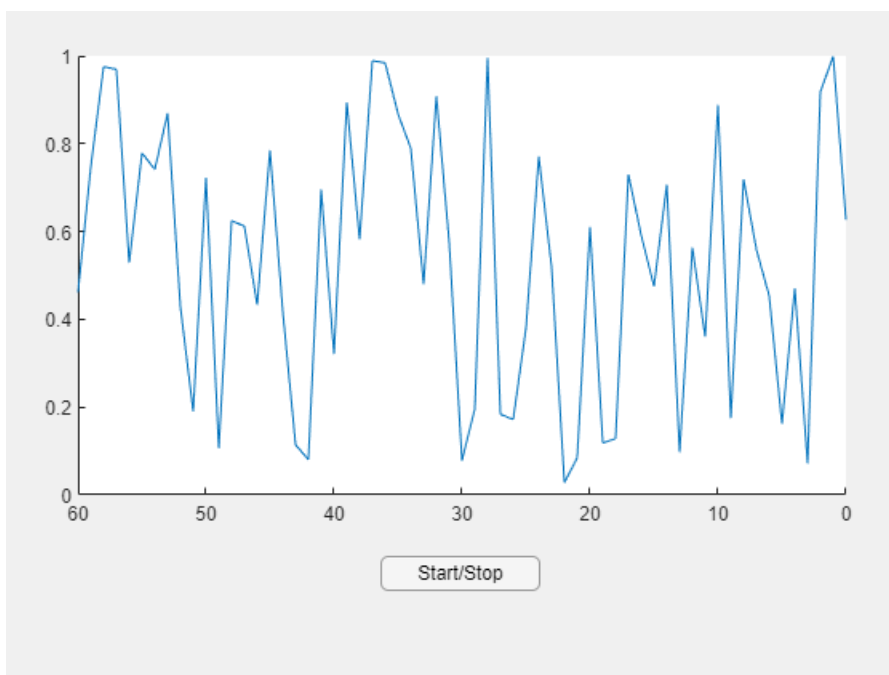
% State button callback function
function StateButtonChanged(obj,~)
    switch obj.Value
        case 0
            stop(RandTimer);
        case 1
            % If timer is not running, start it
            if strcmp(RandTimer.Running,"off")
                start(RandTimer);
            end
    end
end
```

```

        end
    end
end

% Figure close request function
function CloseRequest(~,~)
    % Stop timer, then delete timer and figure
    stop(RandTimer);
    delete(RandTimer);
    delete(uif);
end
end

```



## Plots in Apps: Improved responsiveness of axes interactions within apps

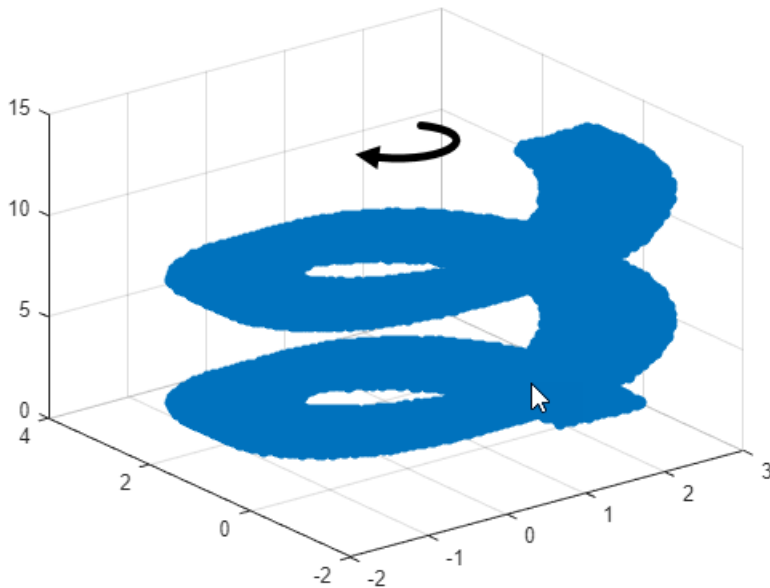
Responsiveness is improved for panning, rotating, and zooming into a region of interest within a plot. To observe the improvement, the plot must be created in an app or in a figure created with the `uifigure` function. The improvement is most noticeable for plots with large numbers of points, or those that involve complex effects such as lighting, transparency, or texture maps. Systems equipped with modern GPUs are more likely to show the improvement.

For example, create a 3-D scatter plot with 200,000 points. If you run this code on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system with an NVIDIA Quadro® P620 GPU, and then drag to rotate the plot, the rotation is smoother and responds more quickly to the drag gesture than in the previous release.

```

z = linspace(0,4*pi,200000);
x = 2*cos(z) + rand(1,200000);
y = 2*sin(z) + rand(1,200000);
f = uifigure; a = axes(f);
scatter3(a,x,y,z,"filled")

```

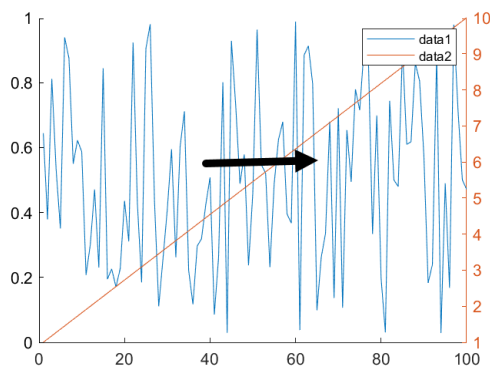


## Plots in Apps: Improved responsiveness of axes interactions in plots with two y-axes

Responsiveness is improved for panning, rotating, and zooming into a region of interest within a UIAxes or Axes object in MATLAB Online that contains a chart created with `yyaxis` and a legend. For such charts, updates are faster, and interactions are smoother in R2022a than in the previous release.

For example, create a UIAxes object with two y-axes and a legend. If you run this code on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system, when you drag the cursor to pan the view of a UIAxes object, the axes pan faster and track the cursor more closely in R2022a than in R2021b.

```
f = uifigure;
a = uiaxes(f);
plot(a,1:100,rand(1,100));
yyaxis(a,"right");
plot(a,1:100,linspace(1,10,100));
legend(a);
```





## Plots in Apps: Faster animations in apps when multiple figures are open

Animations show improved performance in apps when multiple figures are open. To observe the improvement, all figures must be created with the `uifigure` function.

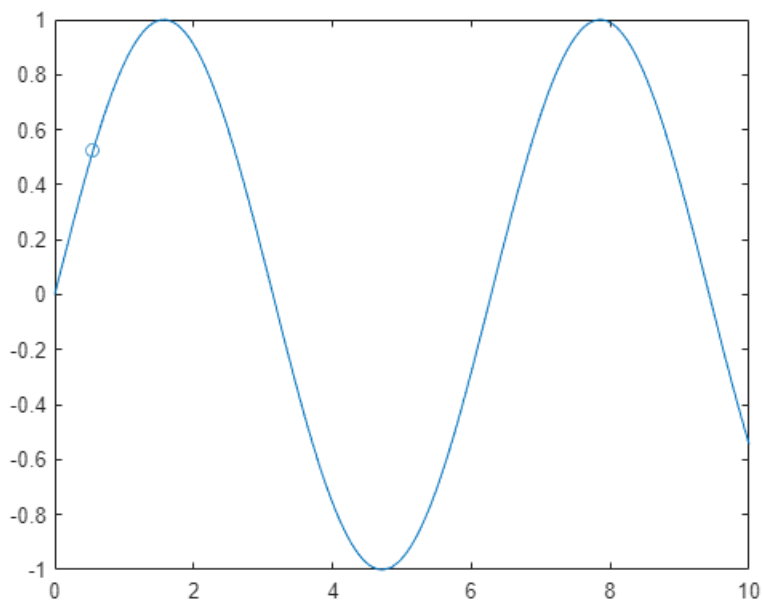
For example, this code opens 10 empty figures and an additional figure containing a plot. The plot displays one marker that traces the path of a line at every iteration of a loop. The loop is about 6x faster than in the previous release.

```
function movingmarker

% Open 10 empty figures
for n = 1:10
    uifigure("Position",[10+n*3 10+n*3 200 200]);
end

% Create a figure and a plot with one marker
uif = uifigure;
x = linspace(0,10,200);
y = sin(x);
ax = axes(uif);
p = plot(ax,x,y,"-o","MarkerIndices",1);

% Move the marker along the sine wave
tic
for idx = 2:200
    p.MarkerIndices = idx;
    drawnow
end
toc
end
```



The approximate execution times for the loop are:

**R2021b:** 32.6 s

**R2022a:** 5.4 s

The code was timed on a Windows 10, Intel Xeon CPU W-2133 @ 3.60 GHz test system by calling the `movingmarker` function.

## **Property Inspector: Improved performance when opening for the first time**

The Property Inspector for a figure window shows improved performance when opening for the first time in a MATLAB session. The delay between clicking the Property Inspector icon or calling `inspect` and the inspector being ready is reduced. The improvement is most noticeable as the plot in the figure window becomes more complex.

For example, open the Property Inspector for a plot of a 5-by-5 matrix by calling `inspect`. If you run this code on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system, you can use the Property Inspector sooner in R2022a than in R2021b.

```
r = rand(5,5);  
p = plot(r);  
inspect(p)
```

## Software Development Tools

### Projects: Reduce test runtime in continuous integration workflows using the dependency cache

You can now specify where your project stores the dependency analysis results. In agile development workflows that use Git™ and a continuous integration (CI) server, share the dependency cache file (.graphml) to run an incremental dependency analysis and reduce the test suite runtime. See “Continuous Integration Using MATLAB Projects and Jenkins”.

To set the project dependency cache file, on the **Project** tab, in the **Environment** section, click **Details**. In **Dependency cache file**, browse to and specify a GraphML file. If the cache file does not exist, the project creates it for you.

Alternatively, you can create and set the project dependency cache programmatically:

```
matlab.project.example.timesTable
proj = currentProject;
proj.DependencyCacheFile = "myProjectCacheFile"
```

### Dependency Analyzer: Save dependency graph as image

You can now save the dependency analysis results as an image. See “Export Dependency Analysis Results”.

### Code Compatibility Analyzer App: Identify and address compatibility issues against current version of MATLAB

The MATLAB Code Compatibility Report is now available as an app. You can access the **Code Compatibility Analyzer** from the apps gallery in MATLAB or from the command line using `codeCompatibilityAnalyzer`. For more information, see “MATLAB Code Compatibility Analyzer”.

### Unit Testing Framework: Create test classes interactively using the Current Folder browser

You can now create a test class using the Current Folder browser in MATLAB and MATLAB Online. To create a new test class, right-click in the Current Folder browser and then select **New > Test Class**.

### Unit Testing Framework: Create temporary folders that are automatically removed

The `matlab.unittest.TestCase` class has a new method `createTemporaryFolder` that creates a temporary folder for your tests. The lifecycle of the folder is tied to the test case. Once the test case goes out of scope, the testing framework removes the folder.

## Unit Testing Framework: Generate DOCX, HTML, and PDF reports after test execution

The `matlab.unittest.TestResult` class has three new methods that enable you to generate various test reports from test results. You can run your tests and collect the test results, and then generate test reports from part or all of your results:

- To generate a DOCX report from the test results, use the `generateDOCXReport` method.
- To generate an HTML report from the test results, use the `generateHTMLReport` method.
- To generate a PDF report from the test results, use the `generatePDFReport` method.

With this feature, you are no longer required to run tests using a `TestReportPlugin` instance. For example, run your tests and then generate an HTML report from the test results. Save the report as `report.html` in a folder named `myResults`.

```
suite = testsuite("MyTestClass");  
runner = testrunner;  
results = run(runner,suite);  
generateHTMLReport(results,"myResults",MainFile="report.html")
```

## Unit Testing Framework: Debug uncaught errors in tests

Starting in R2022a, when a test runner with a `StopOnFailuresPlugin` instance encounters an uncaught error, MATLAB enters debug mode at the source of the error and lets you use debugging commands to investigate the cause of the error. In previous releases, while the plugin stops the test run to report the error, debugging capabilities are limited because the error disrupts the stack.

## Unit Testing Framework: Collect statement and function coverage metrics for your source code

Starting in R2022a, when you generate an HTML code coverage report using the `CoverageReport` format, the report displays statement and function coverage metrics:

- Use statement coverage to see whether every MATLAB statement in your source code is executed at least once.
- Use function coverage to see whether every function in your source code is called at least once.

In previous releases, you can generate only line coverage metrics for your source code. Compared to line coverage, statement and function coverage provide a more detailed analysis of the source code covered by the tests.

## Functionality being removed or changed

### pack function will be removed in a future release

*Warns*

The `pack` function will be removed in a future release. There is no replacement for this function because you do not need to use it on a 64-bit system. For more information about strategies for reducing memory usage, see “Strategies for Efficient Use of Memory” and “Resolve “Out of Memory” Errors”.

## **matlab.unittest.TestSuite.fromFolder includes tests from package folders when creating a test suite**

### *Behavior change*

Starting in R2022a, the `matlab.unittest.TestSuite.fromFolder` method treats folders and packages the same way, and includes tests defined within package folders when creating a test suite. For example, `suite = matlab.unittest.TestSuite.fromFolder(pwd, IncludingSubfolders=true)` creates a suite from all the test files in the current folder and any of its subfolders, including package folders. In previous releases, the method ignores any tests defined in a package folder and its subfolders.

This behavior change also applies to the `testsuite`, `runtests`, and `runperf` functions when they operate on a folder containing tests. With the consistent treatment of folders and packages, creating a suite from all test files within a folder and its subfolders becomes more convenient and independent of the folder structure.

To exclude tests defined within packages, filter the suite being constructed or returned by `fromFolder`. For example, create a filtered test suite comprising tests whose names do not include any dots (that is, do not refer to any packages).

```
import matlab.unittest.TestSuite
import matlab.unittest.selectors.HasName
import matlab.unittest.constraints.ContainsSubstring
suite = TestSuite.fromFolder(pwd, HasName(~ContainsSubstring(".")), ...
    IncludingSubfolders=true);
```

## **builddocsearchdb creates searchable database with new name**

### *Behavior change*

When building a searchable documentation database for custom toolboxes, the `builddocsearchdb` function now creates the subfolder `helpsearch-v4` to contain the search database files. Previously, `builddocsearchdb` created a subfolder named `helpsearch-v3`.

To ensure the documentation for the custom toolbox is searchable in R2022a, run `builddocsearchdb` against your help files using MATLAB R2022a. Maintain the `helpsearch-v4` subfolder containing the search database files created in R2022a and the `helpsearch-v3` subfolder containing the search database files created in previous releases side by side. Then, when you run any MATLAB release, the Help browser automatically uses the appropriate database for searching your documentation.

## External Language Interfaces

### C++ Interface: Array size help text for functions and methods

If a function or method takes a `clib` or MATLAB array, the generated help text displays size information for the argument. For more information, see “Array Size Help for Functions and Methods”.

### C Interface: Build third-party C library interface using `clibgen.generateLibraryDefinition`

Create interfaces for libraries with C files that are built with C compilers using `clibgen.generateLibraryDefinition`. Use this function to get the benefits of publishing an interface as described in “Build MATLAB Interface to C++ Library” instead of calling the `loadlibrary` function described in “Call C from MATLAB”.

For information about using `clibgen.generateLibraryDefinition` with C files, see **Files in Your Library** under the “Tips” section. To build an interface to C libraries, use the `CLinkage` name-value argument.

### C++ Interface: Support for C++ language features

The C++ interface supports these additional C++ language features.

- `std::complex` support for complex scalars and arrays for fundamental types of `double`, `float`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, and `uint64`. For information about mapping these types to MATLAB data, see “Numeric Types”.
- Box array parameters with multiple dimensions. For example, the input to this function is a 2-D array `mat` of size `len-by-typeSz`.

```
void readMatrix2DArr(int const [] mat, size_t len, size_t typeSz)
```

You can specify the SHAPE of the argument as `["len", "typeSz"]`. For more information, see “Define Missing SHAPE Parameter”.

- `std::vector` as data member.

### C++ Interface: Publisher options

The C++ interface supports these build configuration features.

- Specify compiler and linker flags used to build an interface. Use the `AdditionalCompilerFlags` or `AdditionalLinkerFlags` name-value arguments in `clibgen.generateLibraryDefinition` or `clibgen.buildInterface` to pass flags to the compiler and linker. These functions do not validate the flags. The publisher needs to know how the flags affect the build process. For more information, see “Build C++ Library Interface and Review Contents”.
- Build an interface using a specific compiler standard. For example, to build a library defined by `A.hpp` with C++17, type:

```
clibgen.generateLibraryDefinition("A.hpp", AdditionalCompilerFlags="-std=c++17")
```

For more information, see “Specify C++ Compiler Standard”.

- Include static libraries with `.a` file extension on Linux<sup>®</sup> and macOS platforms, and on Windows if the library is compiled with a supported MinGW<sup>®</sup> compiler. To include a static library, use the `Libraries` name-value argument in `clibgen.generateLibraryDefinition` or `clibgen.buildInterface`.

## Call MATLAB from C++: Generate C++ code Interface for MATLAB Packages, Classes, and Functions

The `matlab.engine.typedinterface.generateCPP` creates a C++ header file from MATLAB packages, classes, and functions. For more information, see “What is Strongly Typed Interface for C++?”.

## MATLAB Data Array API: `matlab::data::Array` support for row-major order

The MATLAB data array API supports iterating through the data of a `matlab::data::Array` in either row-major or column-major order. For information about specifying the layout when creating an array, see the `inputLayout` parameter for `createArray`. For information about iterating through an array, see `matlab::data::ColumnMajor`, `matlab::data::ColumnMajorIterator<T>`, `matlab::data::RowMajor`, and `matlab::data::RowMajorIterator<T>`.

## MEX Functions: UTF-8 system encoding on Windows platforms

MATLAB now uses UTF-8 as its system encoding on Windows, completing the adoption of Unicode across all supported platforms. System calls made from within a MEX file take and return UTF-8 encoded strings. If your MEX file contains code or links to third-party libraries that assume a different system encoding, then you might see garbled text and thus need to update the code to be Unicode compliant.

## Python: Use Name=Value syntax to pass keyword arguments to Python functions

You can use MATLAB `Name=Value` syntax as an alternative to the `pyargs` function to pass keyword arguments to Python functions. However, do not mix `Name=Value` arguments with the use of the `pyargs` function.

MATLAB does not support `Name, Value` syntax for passing keyword arguments.

## Python: Convert Python list and tuple types to MATLAB types

You can convert Python `list` and `tuple` types using MATLAB string and numeric converters. For details, see the `py.list` and `py.tuple` entry in the “Explicitly Convert Python Types to MATLAB Types” table. For examples, see “Use Python list Variables in MATLAB” and “Use Python tuple Variables in MATLAB”.

Conversion functions might not work on lists or tuples that contain elements which cannot be converted to the requested type:

```
double(py.list({3.0, 'MATLAB'}))
```

Error using py.list/double  
Conversion of Python element at position 2 to type 'double' failed. All Python elements must be convertible as scalar to the requested type.

Related documentation

For the related documentation, see “Error Converting Elements of list or tuple”.

## Perl 5.34.0: MATLAB support on Windows

As of R2022a, MATLAB on Windows ships with an updated version of Perl, version 5.34.0.

## Version History

If you use the `perl` command on Windows platforms, see <https://www.perl.org/> for information about using this version of the Perl programming language.

## Compilers: Support for Microsoft Visual Studio 2022

As of R2021b Update 3, MATLAB supports Microsoft Visual Studio® 2022 for building C and C++ interfaces, MEX files, and standalone MATLAB engine and MAT-file applications.

## Functionality being removed or changed

### Python: Version 3.7 is no longer supported

*Errors*

Support for Python version 3.7 is discontinued. For continued support for your applications, upgrade to a supported version of Python. For supported version information, see [Versions of Python Compatible with MATLAB Products by Release](#).

### MEX file macro `FORTRAN_COMPLEX_FUNCTIONS_RETURN_VOID` has been removed

*Behavior change*

For MEX files built in R2021b and earlier, MATLAB provided a macro, `FORTRAN_COMPLEX_FUNCTIONS_RETURN_VOID`, to handle platform-dependent calling syntax differences for passing complex numbers to Fortran BLAS and LAPACK functions. As of R2022a, you no longer need a different calling syntax on different platforms, and the macro for handling this difference has been removed.

To update your code, replace statements such as these using `FORTRAN_COMPLEX_FUNCTIONS_RETURN_VOID`:

```
/* Call BLAS function */  
/* Use a different call syntax on different platforms */  
#ifdef FORTRAN_COMPLEX_FUNCTIONS_RETURN_VOID  
    zdotu(&result, &nElements, zinA, &incx, zinB, &incy);  
#else  
    result = zdotu(&nElements, zinA, &incx, zinB, &incy);  
#endif
```

with:



```
/* Call BLAS function */  
zdotu(&result, &nElements, zinA, &incx, zinB, &incy);
```

See `dotProductComplex.c`.

### **NET.addAssembly no longer removes leading spaces for Windows drive letter paths**

#### *Behavior change*

Before R2022a, on Windows platforms, the `NET.addAssembly` function removed leading spaces in paths specifying the drive letter. If the full path to your assembly is invalid with leading spaces, then remove the spaces before calling `NET.addAssembly`.



# R2021b

---

**Version: 9.11**

**New Features**

**Bug Fixes**

**Version History**

## Environment

### Editor Selection: Select and edit a rectangular area of code

In the Editor, you now can select a rectangular area in your code (also known as *column selection* or *block edit*) by pressing the **Alt** key while making a selection with the mouse. On macOS systems, use the **Option** key instead. Selecting and editing a rectangular area of code is useful if you want to copy or delete several columns of data, or if you want to edit multiple lines at one time.



For example, select the second column of data in A.

```
A = [10 20 30 40 50; ...
     60 70 80 90 100; ...
     110 120 130 140 150 ];
```

Type 0 to set all the selected values to 0.

```
A = [10 0| 30 40 50; ...
     60 0| 80 90 100; ...
     110 0| 130 140 150 ];
```

### Editor Display: Zoom in and out in the Editor

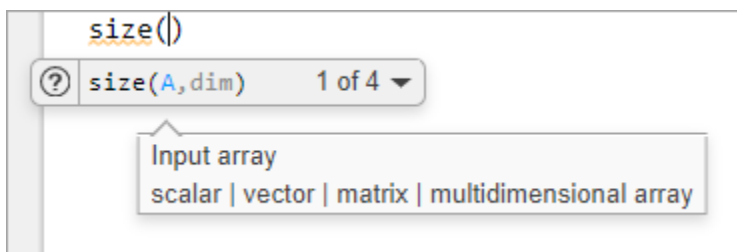
To zoom in or out in the Editor, go to the **View** tab, and in the **Zoom** section, select the  **Zoom In** or  **Zoom Out** button. As you zoom, MATLAB displays the current scale in the bottom-right corner of the Editor. You also can hold the **Ctrl** key and move the scroll wheel, or press **Ctrl+Plus** and **Ctrl+Minus**. On macOS systems, use the **Command** key and move the scroll wheel, or press **Command+Shift+Plus** and **Command+Shift+Minus**.

To return to the default scale, in the **View** tab **Zoom** section, select  **Reset Zoom**. You also can press **Ctrl+Alt+0** (**Command+Alt+0** on macOS).



### Editor Code: Show code suggestions and completions automatically

Starting in R2021b, when you write commands in the Editor, MATLAB automatically displays contextual hints for arguments, property values, and alternative syntaxes. In previous releases, MATLAB only completes names in the Editor after a **Tab** key press.

For example, if you want to use the `size` function, MATLAB automatically displays the syntax information to help you write the command as you type.




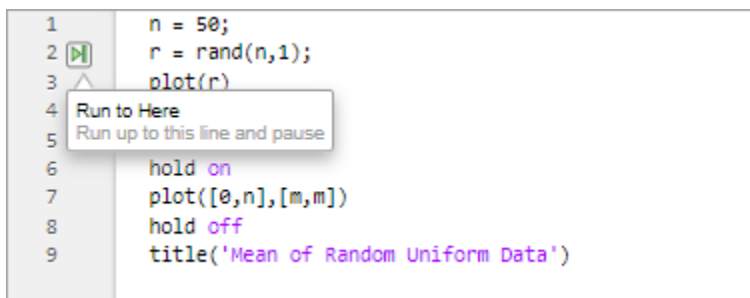
MATLAB also automatically suggests and completes the names of functions, models, MATLAB objects, files, folders, variables, structures, graphics properties, parameters, and options.



You can disable automatic completions in the Editor and Live Editor by having MATLAB suggest and complete names only after you press the **Tab** key. To do so, on the **View** tab, in the **Display** section, click the  **Automatic Completions** button off. You also can go to the **Home** tab, and in the **Environment** section, click  **Preferences**. Then, select **Editor/Debugger > Automatic Completions** and in the **Suggestions and completions** section, select **Show on tab**.



For more information, see [Check Syntax as You Type](#).

## Editor Debugging: Diagnose problems in scripts and functions using inline debugging controls and a breadcrumb-style function call stack

When debugging code in the Editor, you now can diagnose problems using inline debugging controls. For example, to run to a specific line of code and then pause, click the  button to the left of the line.



To step into a file, click the  button directly to the left of the function you want to step into. After stepping in, click the  button at the top of the file to run the rest of the called function, leave the called function, and then pause.


By default, the  button only appears for user-defined functions and scripts. To show the button for MathWorks® functions as well, on the **Home** tab, in the **Environment** section, click  **Preferences**. Then, select **MATLAB > Editor/Debugger**, and in the **Debugging in the Live Editor** section, clear the **Only show Step in button for user-defined functions** option.

When you step into a called function or file, the Editor displays an improved breadcrumb-style list of the functions MATLAB executed before pausing at the current line (also called the function call stack). The function call stack is shown at the top of the file and displays the functions in order, starting on the left with the first called script or function, and ending on the right with the current script or function in which MATLAB is paused.




For more information, see [Debug MATLAB Code Files](#).


## Editor Refactoring: Automatically convert selected code to a function

Break large scripts or functions into smaller pieces by converting selected code into functions in files or local functions. With one or more lines of code selected, on the **Editor** tab, in the **Code** section, click the  **Refactor** button, and then select from the available options. MATLAB creates a function with the selected code and replaces the original code with a call to the newly created function.

## Editor Code: Automatically complete block endings, match delimiters, and wrap comments while editing code

MATLAB now automatically completes parentheses and quotes when you enter code in the Editor. For example, if you type an open parenthesis in the Editor, MATLAB automatically adds the closing parenthesis. MATLAB also automatically completes comments, character vectors, strings, and parentheses split across two lines.

You also can have MATLAB automatically complete block endings. To do so, on the **Home** tab, in the **Environment** section, click  **Preferences**. Select **Editor/Debugger** > **Automatic Completions** and in the **Autocoding options** section, select one or more of the **Autocomplete block endings** options.

To undo an automatic code completion, press **Ctrl+Z** or the  **Undo** button. To disable automatic code completions, in the **Editor/Debugger** > **Automatic Completions** preferences, clear one or more of the options in the **Autocoding options** section.

## Editor Sections: Create sections with an improved appearance

Starting in R2021b, sections in the Editor have an improved appearance. To create a new section, go to the **Editor** tab and in the **Section** section, click the **Section Break** button. The new section is highlighted with a blue border, indicating that it is selected.

```

1  %% Calculate and Plot Sine Wave
2  % Define the range for x.
3  % Calculate and plot y = sin(x).
4  x = 0:1:6*pi;
5  y = sin(x);
6  plot(x,y)
7
8  %% Modify Plot Properties
9  title('Sine Wave')
10 xlabel('x')
11 ylabel('sin(x)')
12 fig = gcf;
13 fig.MenuBar = 'none';

```

To maximize the space available for editing code in the Editor, you can hide the Run to Here and Code Folding margins. This minimizes the gray area to the left of your code. To hide the two margins, right-click the gray area to the left of your code and clear the **Show Run to Here Margin** and **Show Code Folding Margin** options.

```

1  %% Calculate and Plot Sine Wave
2  % Define the range for x.
3  % Calculate and plot y = sin(x).
4  x = 0:1:6*pi;
5  y = sin(x);
6  plot(x,y)
7
8  %% Modify Plot Properties
9  title('Sine Wave')
10 xlabel('x')
11 ylabel('sin(x)')
12 fig = gcf;
13 fig.MenuBar = 'none';

```

As part of this change, the options for changing the appearance of code sections in the Editor have been removed. These options were previously available in the **MATLAB > Colors > Programming Tools** preferences, in the **Section display options** section.

For more information about sections in the Editor, see [Create and Run Sections in Code](#).

## Editor Code: Change the case of text and code

You can change the case of selected text or code in the Editor from all uppercase to lowercase, or vice versa. To change the case, select the text, right-click, and select **Change Case**. You also can press **Ctrl+Shift+A** to change the case. If the text contains both uppercase and lowercase text, MATLAB changes the case to all uppercase.

## Editor Bookmarks: Maintain bookmarks after closing a file

Starting in R2021b, MATLAB maintains all bookmarks after you close a file in the Editor. In previous releases, MATLAB does not maintain bookmarks after closing a file.

For more information, see [Go To Location in File](#).


## Live Editor Controls: Set default values for sliders, drop-down lists, check boxes, and edit fields

You can set the default values for sliders, drop-down lists, check boxes, and edit fields in your live scripts. To set the default value for a control, right-click the control and select **Configure Control**. Then, in the **Defaults** section, specify a default value by entering the value or by selecting a workspace variable from the list. The list shows only valid variables for the control. For drop-down lists, select the default value from the list of items.


To restore the default value for a control, right-click the control and select **Restore Default Value**.

For more information, see [Add Interactive Controls to a Live Script](#).

## Live Editor Animations: Export animations to movies or animated GIFs

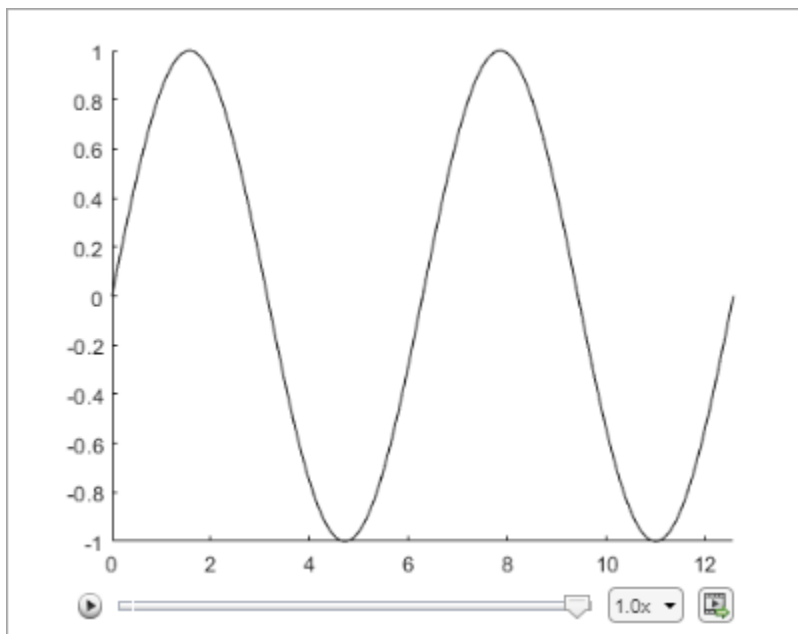
Export animations to movies or animated GIFs using the new  Export Animation button in the Live Editor animation playback controls. The Export Animation button is not supported for animations generated by the `movie` function.

For example, this code animates a line growing as it accumulates 2000 data points in the Live Editor.

When the animation is done playing, playback controls, including the new  Export Animation button, display within the figure window.

```
h = animatedline;
axis([0 4*pi -1 1])
x = linspace(0,4*pi,2000);

for k = 1:length(x)
    y = sin(x(k));
    addpoints(h,x(k),y);
    drawnow
end
```



For more information about creating animations, see [Animation Techniques](#).

## Live Editor Figures: Interact with real MATLAB figures and resize them with improved layouts

Live Editor output figures are now real MATLAB figures with most of the interaction capabilities of standalone MATLAB figures. In addition, when you resize a figure in the Live Editor, the font sizes and spacing between elements in the figure now automatically adjust to provide the best possible presentation for the new size.







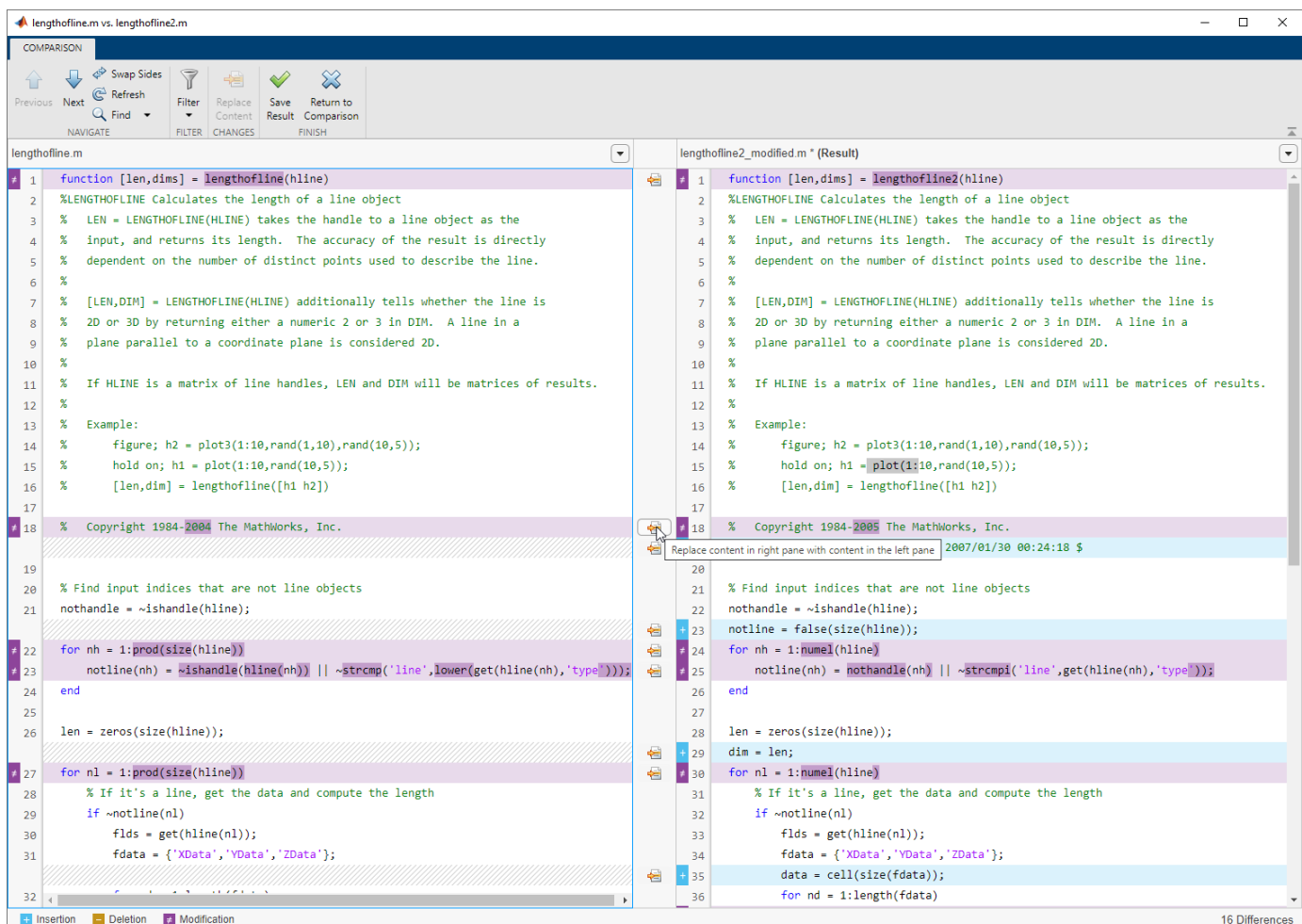
## Live Editor: Improved performance when saving live scripts or functions

Saving live scripts and live functions in the Live Editor is faster in R2021b than in R2021a. The improvement is most noticeable when you save live functions with more than 1000 lines of code and live scripts with fewer than 100 lines of code.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, saving an example live function containing 4000 lines of code takes approximately 2.05 seconds in R2021b. In R2021a, saving the same live function takes approximately 2.57 seconds.

## Comparison Tool: Compare and merge text files with improved usability, appearance, and syntax highlighting

In R2021b, the comparison tool uses MATLAB Editor syntax highlighting. Text changes and merge choices are now easier to understand. Changes are highlighted with strong colors. Modified lines are highlighted and flagged with the comparison icons , , or . Merging line by line is straightforward, and merge choices are flagged with the merge content icon .



## Version History

Starting in R2021b, you no longer can save the comparison report as HTML or specify whether to show only the differences or the entire files.

### Importing Preferences from Previous Releases: MATLAB checks for preferences from R2019b or newer

During start up, MATLAB checks for a preferences folder that matches the current release. If that folder is not found, MATLAB checks for preferences folders going back as far as R2019b. Releases before R2021b continue to check for up to three previous releases.

MATLAB Release	Previous Release Preferences Folders
R2021b	R2021a, R2020b, R2020a, R2019b
R2021a	R2020b, R2020a, R2019b
R2020b	R2020a, R2019b, R2019a

### Display language: MATLAB uses Windows display language settings for selecting desktop language

MATLAB uses the **Windows display language** settings on Windows 10 to control the MATLAB desktop language. The display language you select on Windows changes the default language used by Windows features like settings and file explorer.

Prior to R2021b, MATLAB controlled the desktop language using the Windows locale setting which is managed in the **Region** settings.

For information about Windows locale settings in MATLAB, see [Set Locale on Microsoft Windows Platforms](#).

For information about managing display language settings, refer to your Windows 10 documentation.

## Functionality being removed or changed

### Increment Value and Run Section tool has been removed


The Increment Value and Run Section tool previously available in the Editor has been removed.

To increment a numeric value within a section, use controls in the Live Editor. For example, this code calculates the factorial of the variable  $x$ .

```
x = 5;
y = factorial(x)

y =
    120
```

To interactively change the value of  $x$ , in a live script, replace the value 5 with a numeric slider. By default, MATLAB reruns the current section when the value of the slider changes.

```
x = 5  ;  
y = factorial(x)  
  
y = 120
```

For more information, see [Add Interactive Controls to a Live Script](#).

## Language and Programming

### **cast Function: Consistent output for all syntaxes with the same data type conversion**

The `cast` function now returns consistent output for all syntaxes with the same data type conversion.

For example, starting in R2021b, both `b = cast(fi(1), 'like', sym(1))` and `b = cast(fi(1), 'sym')` return `b = 1` of the `sym` data type. In previous releases, `b = cast(fi(1), 'like', sym(1))` returns `b = 1` of the `sym` data type, but `b = cast(fi(1), 'sym')` throws an error.

### **Run Code in the Background: Use parallel language to run code asynchronously**

You can now run code in the background using `backgroundPool`. When you run code in the background, you can:

- Run other MATLAB code at the same time as a long running calculation
- Create more responsive user interfaces

Use the background pool with the following parallel language:

- `parfeval` and related functionality such as `afterEach` and `afterAll`
- `parallel.pool.DataQueue` and related functionality such as `afterEach`
- `parallel.pool.PollableDataQueue` and related functionality such as `poll`

For more information, see [Background Processing](#).

### **Portable Parallel Code: Share parallel code and seamlessly run in parallel**

You can now run parallel code even if you do not have Parallel Computing Toolbox™. When you run portable parallel code without Parallel Computing Toolbox, you run the code in serial on your machine. When you run this code with Parallel Computing Toolbox, you can automatically scale up and run the code in parallel on your local machine, on a remote cluster, or in the cloud.

The following parallel language features are available for prototyping:

- `parfeval` — Seamlessly run multiple functions at once (since R2021b)
- `parfor` — Seamlessly run for-loops in parallel (since R2008a)

For more information, see [Run Parallel Language in MATLAB](#).

## Compact Display for Classes: Customize display of information about classes when space is limited

Use the `matlab.mixin.CustomCompactDisplayProvider` class to customize how information about your classes is displayed in a container variable—such as a struct, cell array, or table—where space is limited. Options for customization include:

- Displaying partial sets of data
- Adding annotations
- Controlling how and when class names are displayed

For example, an enumeration class of the days of the week, `WeekDays`, can be customized so that when arrays of `WeekDays` members cannot be fully displayed, MATLAB displays the size, the class, and an annotation. When the width of the Command Window is large enough, the full array is shown:

```
myStruc =
    struct with fields:
        prop1: [Monday    Wednesday    Friday    Saturday    Sunday]
```

When the Command Window is not wide enough to display the full array, the size, the class, and an annotation are shown:

```
myStruc =
    struct with fields:
        prop1: 1x5 Weekdays (Enum of days of week)
```

## Class Aliasing: Create aliases for renamed classes to maintain backward compatibility

When you need to change a class name, you can create an alias to preserve compatibility with code written before the name change. The `matlab.alias.AliasFileManager` class provides an API for defining and implementing aliases. Once you define an alias, you can use the alias anywhere you use the class name. Aliases maintain backward compatibility when reloading objects with the old name into a MATLAB version that uses the new name, as well as forward compatibility when loading objects saved in a newer version into an older version that predates the definition of the alias. MATLAB substitutes the new class name whenever it encounters the old name.

## Modular Indexing: Customize class indexing operations individually using new superclasses

To customize how indexing operations behave with your class in previous versions of MATLAB, you need to overload the `subref` and `subsasgn` methods. Doing so requires implementing code for all indexing reference and assignment operations, including parentheses, dot, and brace operations, even if you want to change only one type of indexing operation.

Starting in R2021b, you can inherit from three new superclasses to customize parentheses, dot, and brace indexing operations individually:

- `matlab.mixin.indexing.RedefinesParen` - Customize parentheses reference, assignment, and deletion operations.
- `matlab.mixin.indexing.RedefinesDot` - Customize dot reference and assignment operations.
- `matlab.mixin.indexing.RedefinesBrace` - Customize brace reference and assignment operations.

You can inherit from one or more of these classes without affecting how the other indexing operations work. These classes also enable you to forward levels of indexing in compound statements to other MATLAB values. For example, your class can implement custom parentheses indexing for the first level of a compound reference and then allow MATLAB to apply the other levels to an object contained by your class.

## Scalar Classes: Inherit from the `matlab.mixin.Scalar` superclass to ensure instances behave as scalars

Instances of classes that inherit from `matlab.mixin.Scalar` must behave as scalars. You cannot form arrays of instances of such a class, including empty arrays, and you cannot concatenate instances. This class is useful for cases in which concatenation does not make sense, like a dictionary or other container class in which parentheses indexing is customized to access data in the class, not to form or access arrays.

## startat Function: Time zone information in datetime objects now supported

The `startat` function now recognizes time zone and daylight savings time information of datetime inputs.

## Functionality being removed or changed

### **newclass input argument of the syntax `cast(A,newclass)` is now case-sensitive**

*Behavior change*

Starting in R2021b, the `newclass` input argument of the syntax `cast(A,newclass)` is case-sensitive. You must specify `newclass` as a character vector or a string of lowercase letters that represents the new data type.

For example, to convert a `double` value to the `int8` data type, you must use `b = cast(1.234,'int8')`. The function syntax `b = cast(1.234,'Int8')` now throws an error.

### **Defining classes and packages: Using `schema.m` will not be supported in a future release**

*Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

### **InexactCaseMatch and InexactCaseMatchForExtension Errors: These errors are replaced by UndefinedFunction error**

*Behavior change*

The `InexactCaseMatch` and `InexactCaseMatchForExtension` errors have been removed and MATLAB throws an `UndefinedFunction` error instead. This change does not generate new errors in code that did not previously throw an error.

## Data Analysis

### **Compute by Group Live Editor Task: Interactively summarize, transform, or filter groups of data**

Use the **Compute by Group** Live Editor task to interactively compute statistics, transform data, or filter data by group. To open the task in the Live Editor, use the **Task** menu on the **Live Editor** tab.

### **Normalize Data Live Editor Task: Interactively center and scale data**

Use the **Normalize Data** Live Editor task to visualize the effects of centering and scaling data using various methods, such as the z-score. To open the task in the Live Editor, use the **Task** menu on the **Live Editor** tab.

### **Clean Missing Data Live Editor Task: Define missing values**

When using the **Clean Missing Data** Live Editor task, you can now define missing value indicators that are different from the standard MATLAB missing values.

### **trenddecomp Function: Find trends in data**

Use the `trenddecomp` function to additively decompose data into a long-term trend and seasonal trends.

### **min and max Functions: Specify the comparison method for determining minimum and maximum values**

The `min` and `max` functions now accept the `'ComparisonMethod'` parameter, which specifies a method for determining the minimum and maximum values of the input while preserving the sign in the output.

### **unique\_tol Function: Options to control element selection and preserve range of data**

`unique_tol` has two new options to control behavior:

- `occurrence` argument: Specify whether the algorithm begins with the highest or lowest elements in the input data. This can change which element, among several that are within tolerance of each other, is selected as being unique. The default is to begin with the lowest elements.
- `'PreserveRange'` name-value argument: Specify whether the range of the output data should be the same as the input data.

### **Data Preprocessing Functions: Specify table variable as sample points vector**

When you operate on table input data, the following functions now allow you to specify which variable in the table to use with the `'SamplePoints'` parameter:



- `fillmissing`
- `filloutliers`
- `ischange`
- `islocalmax`
- `islocalmin`
- `isoutlier`
- `rmoutliers`
- `smoothdata`

### **dateshift Function: Shift to next occurrence of weekday or weekend day**

You can now use the `'weekday'` and `'weekend'` arguments to shift the elements of a `datetime` array when using the `dateshift` function.

- To shift to the next occurrence of a weekday on or after each element of the input `datetime` array, use `'weekday'`.
- To shift to the next occurrence of a weekend day on or after each element of the input `datetime` array, use `'weekend'`.

### **isbetween Function: Support for open, closed, and half open intervals**

The `isbetween` function now supports open, closed, and half open intervals. In previous releases, `isbetween` supports only closed intervals.

### **isregular Function: Support for datetime and duration data types**

You can now use the `isregular` function to determine if a timetable, `datetime` vector, or duration vector is regular. In previous releases, you can use `isregular` only on a timetable.

### **istabular Function: Determine if input is a table or timetable**

To determine if an input variable is either a table or a timetable, use the `istabular` function.

Using this function is equivalent to using the statement `tf = istable(A) || istimetable(A)`, but is more convenient.

### **retime and synchronize Functions: Median and mode methods supported**

When you synchronize data in timetables, you can now specify `'median'` and `'mode'` as aggregation functions. For more information, see `retime` and `synchronize`.

### **timeofday Function: Return the date as the second output argument**

You can now return the dates from the elements of a `datetime` array as the second output argument from the `timeofday` function.

## **timeseries2timetable Function: Convert timeseries objects to timetables**

To convert `timeseries` objects to timetables, use the `timeseries2timetable` function.

### **Functionality being removed or changed**

#### **isordinal accepts input argument that has any data type**

*Behavior change*

The `isordinal` function now accepts an input argument that has any data type. In previous releases, `isordinal` threw an error if the input argument was not a categorical array.

#### **Synchronize Timetables Live Editor task synchronizes an unlimited number of timetables**

*Behavior change*

The **Synchronize Timetables** Live Editor task can now synchronize an unlimited number of timetables. In previous releases, the task can synchronize no more than five timetables.

#### **timeseries2timetable replaces ts2timetable**

*Behavior change*

The `timeseries2timetable` function replaces the `ts2timetable` function, although `ts2timetable` is still provided. The two functions are synonyms. In R2021a, MATLAB provides `ts2timetable` only.

## Data Import and Export

### sftp Function: Connect to SFTP servers

MATLAB can connect to SFTP servers for encrypted data transfers. Create an SFTP connection object using the `sftp` function to read data from an SFTP server.

### Datstores: Specify FileSet objects as data locations for some datstores

Some datastore functions and objects accept `FileSet` objects as the locations of files to include in the datastore. `FileSet` objects provide increased performance compared to file paths or `DsFileSet` objects. This functionality is supported by these functions:

- `tabularTextDatastore`
- `spreadsheetDatastore`
- `fileDatastore`
- `keyValueDatastore`
- `talDatastore`
- `parquetDatastore`
- `imageDatastore`
- `signalDatastore` (Signal Processing Toolbox)
- `audioDatastore` (Audio Toolbox)
- `mdfDatastore` (Vehicle Network Toolbox)

### Table Import: Read tables from HTML and Microsoft Word documents

The `readtable` function now supports reading tables from HTML and Microsoft Word files.

To customize import options for HTML and Microsoft Word files, use `htmlImportOptions` and `wordDocumentImportOptions`, respectively. To automatically detect import options from files, use the `detectImportOptions` function.

### HDF5 Interface: Use new functionality in support of HDF5 1.10.7

Use these new capabilities of the MATLAB HDF5 function interfaces:

- Single-Writer/Multiple-Reader (SWMR) — Write data to an HDF5 file in one process while you concurrently read from the file in one or more reader processes. For more information, see [Read and Write Data Concurrently Using Single-Writer/Multiple-Reader \(SWMR\)](#).
- Virtual Dataset (VDS) — Use the MATLAB low-level interface to access data stored across multiple HDF5 files, including files in remote locations, as a single, unified HDF5 dataset. You can also read data stored in Virtual Datasets using the HDF5 high-level interface. For more information, see [Work with HDF5 Virtual Datasets \(VDS\)](#).
- Metadata Cache Fine-Tuning — Improve performance by controlling the parameters of the metadata cache, such as limiting the number of file reading attempts.

- Partial Edge Chunk — Control whether to filter partial edge chunks.

## NetCDF Interface: Read and write NC\_STRING data

You can now use the existing high-level and low-level functions to read NC\_STRING data from NetCDF-4 files and write text data as type NC\_STRING.

For more information on data type mapping between the NetCDF API and MATLAB, see Map NetCDF API Syntax to MATLAB Syntax.

## Scientific File Format Libraries: HDF5 and NetCDF libraries are upgraded

The HDF5 library is upgraded to version 1.10.7, and the NetCDF library is upgraded to version 4.7.4.

## Audio, Video, and Image I/O Functions: Run functions in a thread-based environment

You can now run the following functions in the background using MATLAB backgroundPool:

- audioread
- audiowrite
- imwrite
- VideoReader
- VideoWriter

For more information, see Run MATLAB Functions in Thread-Based Environment.

## Image File Format Libraries: LibTIFF library upgraded to version 4.2.0

The LibTIFF library is upgraded to version 4.2.0.

## New Serial Explorer and TCP/IP Explorer apps

Two new apps offer functionality for communicating with your device, instrument, or server:

- The **Serial Explorer** app provides a user interface to connect to and communicate with a serial port device on your machine.
- The **TCP/IP Explorer** app provides a user interface to create a TCP/IP client that communicates with a TCP/IP server.

Launch these apps from the **Apps** tab, under the **Test and Measurement** section. You can also call the serialExplorer and tcpipExplorer commands in the Command Window.

You can use the apps to perform the following operations on your serial port device or TCP/IP client.

- Configure connection and communication properties.
- Write binary or string data.

- Read binary or string data.
- Plot data in a separate figure window.
- Analyze data by viewing it in the **Signal Analyzer** app.
- Export data to the MATLAB workspace.
- Generate a MATLAB script for app interactions that uses the `serialport` or `tcpclient` interface.

For more information about these apps, see **Serial Explorer** and **TCP/IP Explorer**.

## Functionality being removed or changed

### **Video and Image I/O Functions: Pixel value differences might exist between JPEG 2000 images in R2021b and previous versions of MATLAB**

*Behavior change*

In R2021b, when you use the `imread`, `imwrite`, `VideoReader`, or `VideoWriter` functions to read or write JPEG 2000 image files, the image you import or export in R2021b might have pixel value differences with the same image in previous versions of MATLAB.

### **HDF5 Interface: Linux users need to rebuild filter plugins using MATLAB HDF5 1.10.7 shared library**

*Behavior change*

Starting in R2021b, in certain cases, Linux users using a filter plugin with callbacks to core HDF5 library functions need to rebuild the plugin using the shipping MATLAB HDF5 1.10.7 shared library, `/matlab/bin/glnxa64/libhdf5.so.103.3.0`. If you do not rebuild the plugin using this version of the shared library, you might experience issues ranging from undefined behavior to crashes. For more information, see [Build HDF5 Filter Plugins on Linux Using MATLAB HDF5 Shared Library or GNU Export Map](#).

### **ftp Function: FTPClientConfig class, properties, and methods are no longer supported**

The `ftp` function no longer supports the Apache™ `FTPClientConfig` class or any associated objects, properties, or methods. To customize how to parse the `LIST` command output of the FTP server use the `ftp` function's `DirParserFcn` name-value argument.

### **MATLAB Variable Editor: timeseries will no longer be supported in a future release**

*Still runs*

Viewing `timeseries` objects using the MATLAB Variable Editor will no longer be supported in a future release. To view time-indexed data in the Variable Editor, use `timetable` instead.

## Mathematics

### **ode78 and ode89 Functions: High-order Runge-Kutta solvers for ordinary differential equations**

The MATLAB ODE suite has been expanded with two new solvers:

- `ode78` uses 7th- and 8th-order Runge-Kutta formulas
- `ode89` uses 8th- and 9th-order Runge-Kutta formulas

The new solvers expand on the existing Runge-Kutta solvers `ode23` and `ode45`. In particular, `ode78` and `ode89` can be more efficient than `ode45` on nonstiff problems that are smooth, and `ode89` can be more efficient than `ode78` on very smooth problems, when you integrate over long time intervals or when tolerances are tight.

### **pagesvd Function: Perform singular value decomposition on pages of N-D arrays**

Use the `pagesvd` function to perform batched singular value decompositions on the pages of N-D arrays. In this context, the N-D array is treated as a container for several 2-D matrices.

### **svd Function: Option to control output format of singular values**

`svd` has a new option `outputFormat` to control whether the singular values are returned as a vector or diagonal matrix.

### **mpower Function: Improved algorithm for defective matrices**

The `mpower` function has an improved algorithm to handle defective matrices raised to a real power. In previous releases, `mpower` uses an algorithm based on eigenvalue decomposition for these inputs that can return incorrect results for defective matrices. The new algorithm for defective matrices is instead based on the Schur decomposition.

## **Functionality being removed or changed**

### **svd, eig, cond, and pinv functions return NaN for nonfinite inputs**

*Behavior change*

In R2021b, the `svd`, `eig`, `cond`, and `pinv` functions return NaN values when the input contains nonfinite values (`Inf` or `NaN`).

In previous releases, these functions throw an error when the input contains nonfinite values.

## Graphics

### Plotting Table Data: Create scatter plots, bubble charts, and swarm charts by passing tables directly to plotting functions

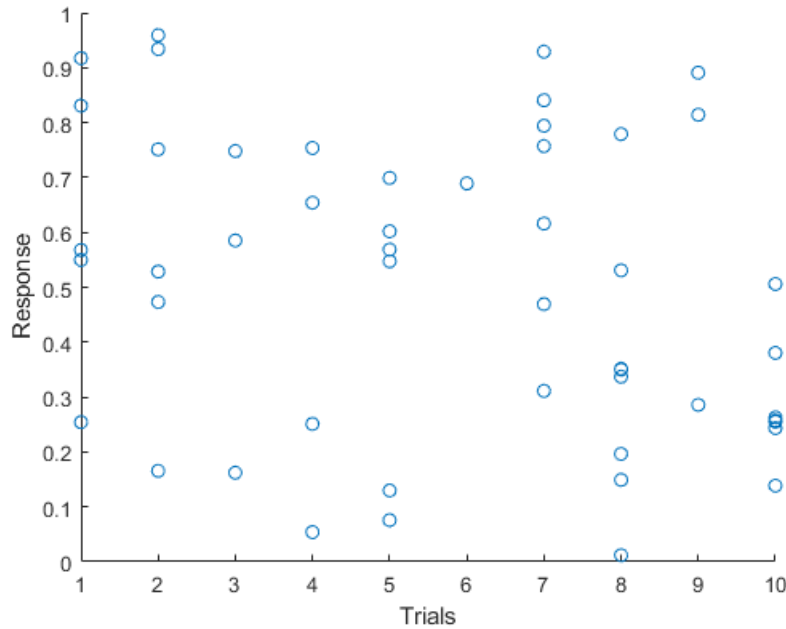
Create plots by passing a table directly to any of these functions: `scatter`, `scatter3`, `bubblechart`, `bubblechart3`, `swarmchart`, `swarmchart3`, `polarscatter`, and `polarbubblechart`. When you specify your data as a table, the axis labels and the legend (if present) are automatically labeled using the table variable names.

The objects returned by these functions have new properties to support tables.

Property	Description
SourceTable	Table containing the data to plot
XVariable, YVariable, and ZVariable	Table variables containing the x, y, and z values for Cartesian plots
ThetaVariable and RVariable	Table variables containing the angle and radius values for polar plots
SizeVariable	Table variable containing the marker size data
ColorVariable	Table variable containing the marker color data
AlphaVariable	Table variable containing the marker transparency data

For example, create a table with the variables "Trials" and "Response". Pass the table to the `scatter` function as the first argument, and indicate the variables you want to plot by name.

```
Trials = randi(10,50,1);
Response = rand(50,1);
t = table(Trials,Response);
scatter(t,"Trials","Response")
```



## Axes Ticks and Colors: Control the appearance of axis tick marks and tick label colors

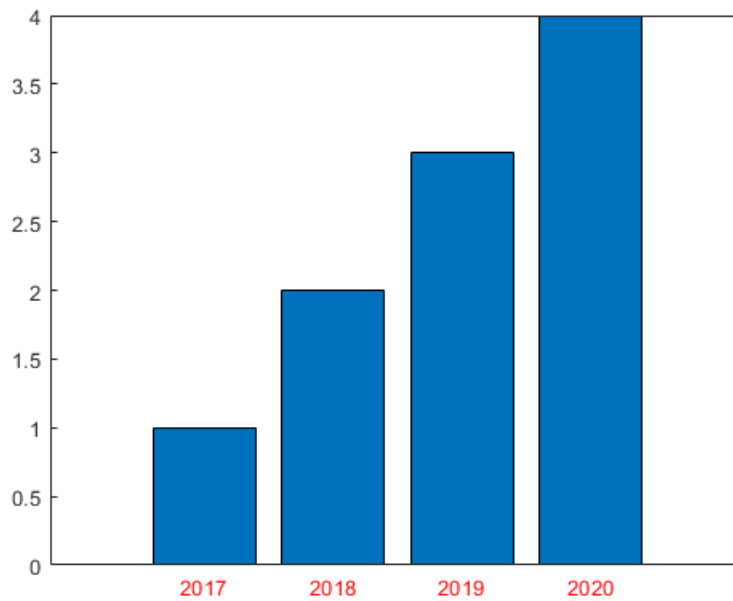
Now, you can remove tick marks and customize tick label colors independently of other elements in the axes.

- **Removing Tick Marks** — Remove all the tick marks from an axes, polar axes, or geographic axes object by setting the `TickDir` property to `'none'`. To remove the tick marks from a specific axis, for example the x-axis, set the `TickDirection` property of the ruler to `'none'`.
- **Customizing Tick Label Colors** — Customize the color of the tick labels on an axis by setting the `TickLabelColor` property of the corresponding ruler object. You can customize tick label colors for any axes, polar axes, or geographic axes.

For example, create a bar chart, and then get the current axes. Remove the x-axis tick marks and change the color of the x-axis tick labels to red.

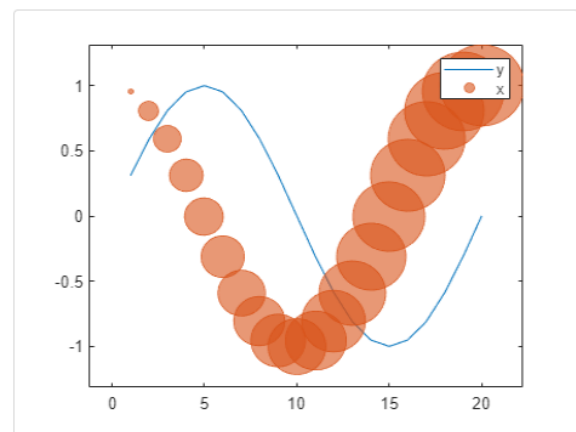
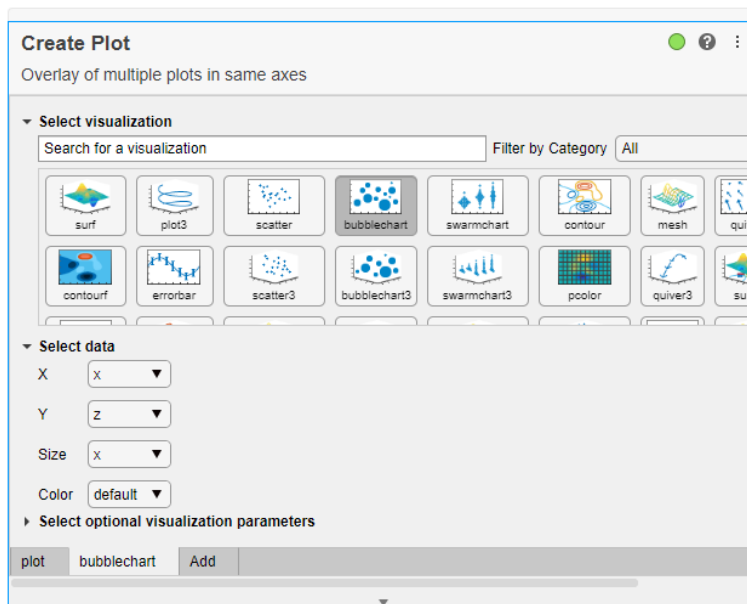
```
bar([2017 2018 2019 2020],1:4)
ax = gca;
ax.XAxis.TickDirection = 'none';
ax.XAxis.TickLabelColor = 'r';
```





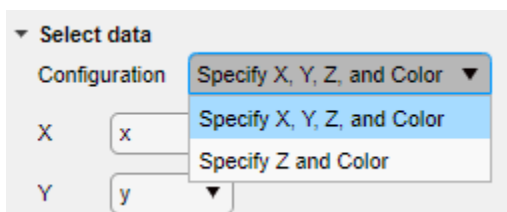
## Create Plot Live Task: Add additional visualizations to generated plots

Easily add additional visualizations to plots generated using the Create Plot Live Task. To add a new plot, click the **Add** tab at the bottom of the Live Task Panel and select the visualization and data. This Live Task combines the plot using the hold function.



## Create Plot Live Task: Control chart input syntax using configuration drop-down

The Create Plot Live Task now supports multiple configurations of charting functions with multiple input syntaxes, including surf and mesh. Use the **Configurations** drop-down menu to select the desired configuration.



## exportgraphics Function: Capture and append graphics to existing PDFs

Capture and append graphics to an existing PDF file by calling the `exportgraphics` function and setting the 'Append' name-value argument to `true`. For example, create a plot and export it as a PDF called 'mycharts.pdf'. Then, create a bar chart and append it to the end of 'mycharts.pdf'. The resulting PDF file has two pages. The plots appear in the PDF in the order that you export them.

```
plot([0 3 1 6 4 10])
exportgraphics(gca, 'mycharts.pdf')
bar([10 20 30 40])
exportgraphics(gca, 'mycharts.pdf', 'Append', true)
```

## stackedplot Function: Support for semilog y-axes

You can create plots using the `stackedplot` function where individual y-axes can be plotted on a log scale. To set a log scale for the y-axis of a plot, set the `YScale` property of the `StackedAxesProperties` object associated with the plot. For more information, see `StackedAxesProperties` Properties.

## Text Objects: Use editInteractions in the Interactions property to click or tap on text to edit

Click or tap to edit text when the `Interactions` property has the value `editInteraction`. The edit interaction is default behavior for title, subtitle, xlabel, ylabel, and zlabel text objects for axes, geographic axes, and polar axes.

## dataTipTextRow Function: Customize data tip content using data properties, such as UserData

You can now assign information to the `DataTipTemplate` property by passing it to the `dataTipTextRow` function as a property name, such as `UserData`.

```
p = patch;
p.UserData = p.XData;
p.DataTipTemplate.DataTipRows(3) = dataTipTextRow('XDataAsUserData', 'UserData');
```

## MATLAB Online™ Accessibility: Use a screen reader to interact with figures

In MATLAB Online™, you can use a screen reader and keyboard commands to pan, zoom, and rotate when you work with plotted data. Using a screen reader is not supported in the Live Editor.

For more information, see [Use a Screen Reader in MATLAB Online](#).

For more details on interacting with MATLAB figures, see [Control Chart Interactivity](#).

## Functionality being removed or changed

### The print options `-opengl` and `-painters` are not recommended

*Still runs*

The following print options are no longer recommended. There are no plans to remove the values, and they will continue to behave the same way as in previous releases. The following table lists the recommended replacement options.

Not Recommended	Replacement Option
The <code>-opengl</code> renderer option. For example: <code>print('-opengl', '-dpdf', 'myfigure.pdf')</code>	Use the <code>-image</code> option. For example: <code>print('-image', '-dpdf', 'myfigure.pdf')</code>
The <code>-painters</code> renderer option. For example: <code>print('-painters', '-dpdf', 'myfigure.pdf')</code>	Use the <code>-vector</code> option. For example: <code>print('-vector', '-dpdf', 'myfigure.pdf')</code>

### plottools functions will be removed in a future release

*Still runs*

The `plottools` functions listed below will be removed in a future release. Use `inspect` to launch the Property Inspector instead.

#### plottools functions

- `plottools`
- `showplottool`
- `figurepalette`
- `plotbrowser`
- `propertyeditor`
- `propedit`
- `plottedit` options `'showtoolsmenu'` and `'hidetoolsmenu'`

## App Building

### **UIAlert, Uiconfirm, and Uiprogressdlg Functions: Mark up text and display equations in dialog boxes**

When you create dialog boxes using the `UIAlert`, `Uiconfirm`, and `Uiprogressdlg` functions, enable markup in the dialog box text using the `Interpreter` name-value argument. Specify the interpreter as `'html'`, `'latex'`, `'tex'`, or `'none'`.

### **AddStyle Function: Add styles to nodes and levels in a tree UI component**

Create styles for specific tree nodes or tree node levels in a tree UI component using the `Uistyle` and `AddStyle` functions. For example, you can make the tree nodes at the top level of the tree red with italic font. To get information on applied styles, query the `StyleConfigurations` property of the `Tree` object. To remove a style from a tree, use the `RemoveStyle` function.

### **UITable Function: Set and query table selections programmatically and control table selection options**

You can now configure selection options of table UI components.

- Set and query the table selection using the `Selection` property.
- Specify whether a user can select table cells, rows, or columns using the `SelectionType` property.
- Specify whether a user can select single or multiple table elements using the `Multiselect` property.
- Update your app whenever a user selects table data by specifying a `SelectionChangedFcn` callback.

Selection options in table UI components are supported only in App Designer apps and in figures created with the `UIfigure` function.

For more information, see [Table Properties](#).

### **UITextarea Function: Program apps to respond while a user is typing in a text area component**

You can now specify a `ValueChangingFcn` callback for a `TextArea` component. The component executes the callback function repeatedly while a user types in the text area.

For more information, see [TextArea Properties](#).

### **Run Code in the Background: Use parallel language to create more responsive apps**

You can now create apps that remain responsive while performing calculations in the background by using `backgroundPool`.

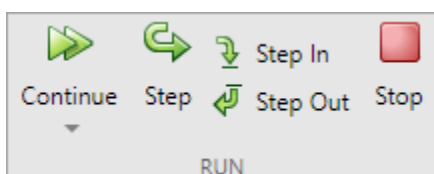
Use the background pool with the following parallel language:




- `parfeval` and related functionality such as `afterEach` and `afterAll`
- `parallel.pool.DataQueue` and related functionality such as `afterEach`

For more information, see [Use the Background to Make Your Apps More Responsive](#).

## App Designer: Debug code in Code View

When debugging code in App Designer, you now can diagnose problems using debugging controls in **Code View**. You can use the controls in the **Run** section of the **Editor** tab to run to the next breakpoint, run the next line of code, or step into or out of a function.




You can also debug your app code using inline debugging controls. For example, to run to a specific line of code and then pause, click the  button to the left of the line. To step into a function, click the  button directly to the left of the function you want to step into. After stepping in, click the  button at the top of the file to run the rest of the called function, leave the called function, and then pause.

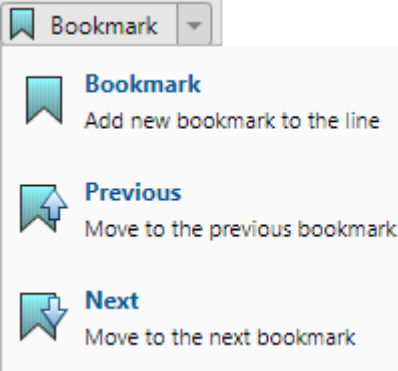
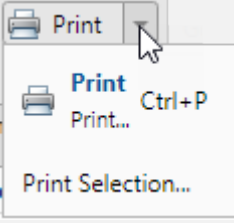
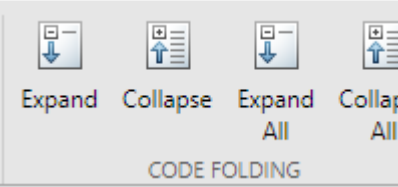
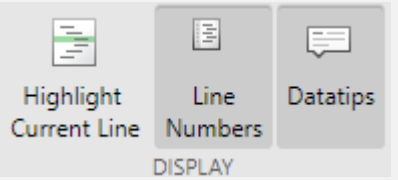
When you step into a called function or file, App Designer displays a breadcrumb-style list of the functions MATLAB executed before pausing at the current line (also called the function call stack). The function call stack is shown at the top of the file and displays the functions in order, starting on the left with the first called script or function, and ending on the right with the current script or function in which MATLAB is paused.



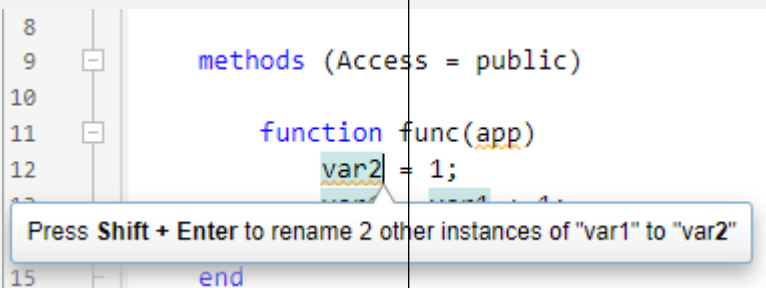
## App Designer: Efficiently manage your app code with tools and shortcuts from Live Editor

Many of the tools and shortcuts for navigating and organizing code that are available in the Live Editor can now be used in App Designer **Code View**. This table lists the functionality that is new to App Designer.

Functionality	Menu Item	Keyboard Shortcut
Wrap comment	Right-click the comment and select <b>Wrap Comments</b> , or in the <b>Editor</b> tab, in the <b>Code</b> section, click the  button.	<b>Ctrl+J</b>
Navigate code using bookmarks	In the <b>Editor</b> tab, in the <b>Navigate</b> section, click <b>Bookmark</b> . Then, select	Set or clear bookmark: <b>Ctrl+F2</b>

Functionality	Menu Item	Keyboard Shortcut
	<p><b>Bookmark</b> to set or clear a bookmark on the current line, or select <b>Previous</b> or <b>Next</b> to navigate between the existing bookmarks.</p> 	<p>Move to previous bookmark: <b>Shift+F2</b></p> <p>Move to next bookmark <b>F2</b></p> <p><b>Ctrl+F2</b></p> <p><b>Shift+F2</b></p> <p><b>F2</b></p>
Print app code	<p>In the <b>Editor</b> tab, in the <b>File</b> section, click <b>Print</b>. You can print the entire document or the current selection.</p> 	<b>Ctrl+P</b>
Fold and expand code	<p>In the <b>View</b> tab, click the buttons in the <b>Code Folding</b> section.</p> 	<p>Expand current fold: <b>Ctrl+Shift+Period (.)</b></p> <p>Collapse current fold: <b>Ctrl+Period (.)</b></p> <p>Expand all folds: <b>Ctrl+Shift+Comma (,)</b></p> <p>Collapse all folds: <b>Ctrl+Comma (,)</b></p>
Toggle display preferences	<p>In the <b>View</b> tab, in the <b>Display</b> section, toggle line highlighting and line numbers.</p> 	N/A


Functionality	Menu Item	Keyboard Shortcut
Duplicate line	Right-click a line and select <b>Duplicate Line(s)</b> .	<b>Ctrl+Shift+C</b>
Insert section break	Right-click a line and select <b>Section Break</b> .	<b>Ctrl+Alt+Enter</b> , or type %%
Convert text to uppercase or lowercase	Highlight the text, right-click it, and select <b>Change Case</b> .	<b>Ctrl+Shift+A</b>
Variable rename	When you rename a variable, App Designer gives you the option to automatically update all other instances of the variable in your code.	<b>Shift+Enter</b>




## Version History

Code folding in App Designer persists even after you close and then reopen the file. In R2021a and earlier releases, when opening a file, App Designer expands all the code.

## App Designer: Interactively modify canvas zoom level and fit canvas to view

In **Design View**, use the zoom controls in the lower right corner of the App Designer canvas, indicated by the  button, to modify the canvas zoom level.

To automatically zoom to fit the entire app in the view, press **Space**. Alternatively, click  **Fit to View** in the **Zoom** section of the **View** tab, or right-click on the canvas and select **Zoom > Fit to View**.

## App Designer: Convert between similar UI components


To convert one type of UI component to another with similar functionality, right-click the component on the canvas or in the **Component Browser** and select **Replace With**. Then, select the component to convert to. Replacing one component with another preserves relevant property values, such as font properties and callbacks that exist for both components. You can convert component types within each of these families:

- Numeric edit field, spinner, slider, and knob

- Edit field and text area
- Label and hyperlink

## App Designer: Add help text for your app

You can now provide help for apps that you create. Help text appears in the Command Window when an app user calls the `help` function and specifies the name of the app.

To add help text, in the **Editor** tab in **Code View**, click  **App Help Text**. Use the App Help Text dialog box to specify the app summary and detailed explanation.

In addition, when an app user views the documentation for your app (for example, by calling the `doc` function or by clicking the documentation link in the help text for the app), the documentation page now displays additional information:

- The top of the page displays the app summary and detailed explanation.
- The Methods Summary section displays the public functions. For each function, it also displays any comment that is inserted after the function definition statement.

## App Designer: Remove auto-reflow behavior from an app with auto-reflow

To convert an app with auto-reflow to an app without auto-reflow, in the **Canvas** tab in **Design View**, click **Convert**. Select the **App without Auto-Reflow** option. Doing so creates a duplicate of your app with the auto-reflow behavior removed.

For more information, see Apps with Auto-Reflow.

## Deployed Web Apps: Deploy web apps directly to the MATLAB Web App Server from within App Designer

Once you have MATLAB Compiler™ installed on the system running MATLAB, package your MATLAB app into a web app from within App Designer by clicking **Share** in the **Designer** tab and selecting **Web App**. In the packaging dialog, specify the server URL to directly deploy your web app to the server once packaging is complete. Authentication must be enabled on the server for this to work. For details, see Authentication (MATLAB Web App Server).

## App Testing Framework: Perform press gestures on axes and UI axes with different selection types

The app testing framework now supports mouse selection types in press gestures that are performed on axes and UI axes. For example, create an axes with a plot and then test a double-click gesture at the point (3, 2).

```
f = uifigure;
ax = axes(f);
plot(ax,1:10)
tc = matlab.uitest.TestCase.forInteractiveUse;
tc.press(ax,[3 2], 'SelectionType', 'open')
```



## App Testing Framework: Perform drag gestures on axes and figures with different selection types

Starting in R2021b, the app testing framework supports drag gestures on UI figures. Additionally, when you test a drag gesture on an axes, UI axes, or UI figure, you can specify the mouse selection type. For example, create a figure and drag on it from the point (100, 200) to the point (200, 300) using a right-click gesture.

```
f = uifigure;
tc = matlab.uitest.TestCase.forInteractiveUse;
tc.drag(f,[100 200],[200 300],'SelectionType','alt')
```

## App Testing Framework: Use any units of measurement in gestures at the center of components

Starting in R2021b, when you perform a gesture at the center of a component, the component or its parent containers can use any units of measurement. In previous releases, the framework does not support containers that use nonpixel units.

For example, create a figure and set its `Units` property to `'normalized'`. Then, create a panel in the figure and press at the center of the panel.

```
f = uifigure;
f.Units = 'normalized';
p = uipanel(f);
tc = matlab.uitest.TestCase.forInteractiveUse;
tc.press(p)
```

If you perform a gesture at the center of a component using a syntax that accepts location as an input (for instance, `press(testcase, comp, location)`), then the figure or parent containers can use only `'pixels'` as their units of measurement.

## Functionality being removed or changed

### Ctrl+Click selects and deselects cells in a table UI component

*Behavior change*

In tables created using the `uitable` function, you can select and deselect noncontiguous table cells by holding **Ctrl** and clicking a cell. In R2021a and earlier releases, **Ctrl+Click** gives focus to a cell and **Shift+Click** selects the cell that has focus.

### App Designer toolstrip organization has changed

*Behavior change*

The organization of the tools in the App Designer toolstrip in **Design View** and **Code View** has changed.

In **Design View**, use the tools in the **Canvas** tab to lay out your app, and use the tools in the **View** tab to manage your **Design View** preferences.

In **Code View**, use the tools in the **Editor** tab to program your app behavior and to run and debug your app, and use the tools in the **View** tab to manage your **Code View** preferences.

### matlab.fonts.editor.codefont.Size setting has been removed

*Errors*


The `matlab.fonts.editor.codefont.Size` setting has been removed. Use the `matlab.fonts.codefont.Size` setting instead. The `matlab.fonts.codefont.Size` setting controls both the App Designer **Code View** font size and the desktop code font size.

To update your code, change instances of the setting `matlab.fonts.editor.codefont.Size` to `matlab.fonts.codefont.Size`. For more information, see `matlab.fonts` Settings.

### **App Designer Smart Indent applies to individual lines**

#### *Behavior change*

When you apply Smart Indent to code in App Designer, the indentation change applies only to the current line. In R2021a and earlier releases, the Smart Indent option applied to the entire document.

To apply Smart Indent to the entire document, in **Code View**, first select all the code (for example, by pressing **Ctrl+A**). Then, apply Smart Indent by clicking the  button in the **Editor** tab, or pressing **Ctrl+I**.

### **CellSelectionCallback property of table UI components is not recommended in uifigure-based apps**

#### *Still runs*

Starting in R2021b, using the `CellSelectionCallback` property to program a response to table selection is not recommended for table UI components in App Designer apps and in figures created with the `uifigure` function. Use the `SelectionChangedFcn` property instead.

To update your code, assign all callback functions assigned to the `CellSelectionCallback` property to the `SelectionChangedFcn` property instead. If a callback function accesses the callback event data, you might need to update the event property names. For example, to access the indices of the elements the user selected, use the `Selection` property of the `TableSelectionChangedData` object. For more information, see `Table` Properties.

## Performance

### table Data Type Indexing: Improved performance when assigning elements by subscripting with curly braces

table subscripted assignment using curly braces is significantly faster in R2021b than in R2021a.

For example, when you assign into three table variables with  $10^6$  elements, performance in R2021b is approximately 4.4x faster, as shown below.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
    indices = randi(1e6,1,10000);

    tic;
    % Assign row vector of random values to randomly chosen row
    for i = indices
        t{i,:} = rand(1,3);
    end
    toc
end
```

The approximate execution times are:

**R2021a:** 7.4 s

**R2021b:** 1.7 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling the `timingTest` function in R2021a and R2021b.

### qrinsert and qrdelete Functions: Improved performance modifying QR factorizations

The `qrinsert` and `qrdelete` functions show improved performance inserting and deleting rows and columns in a QR factorization. The speedup is most noticeable for square matrices of order 1000 or less and is similar in magnitude for both rows and columns.

For example, this code uses a loop to insert and delete columns from the QR factorization of a random 200-by-200 matrix. `qrinsert` and `qrdelete` are about 12x faster than in the previous release.

```
function timingQRMod
X = rand(200);
[Q,R] = qr(X);
y = rand(200,1);
tic
for k = 1:1000
    [Qn,Rn] = qrinsert(Q,R,100,y);
end
toc
tic
for k = 1:1000
    [Qn,Rn] = qrdelete(Q,R,100);
end
```

```
toc  
end
```

The approximate execution times are:

**R2021a:** 1.7 s (insertion) and 1.2 s (deletion)

**R2021b:** 0.15 s (insertion) and 0.10 s (deletion)

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the `timingQRMod` function.

## Titles and Labels in Plots: Improved performance when creating and querying titles or labels in a loop

Creating and querying the following types of titles and labels in a loop has improved performance.

- Plot titles, such as those created with the `title` or `subtitle` functions
- Axis labels, such as those created with the `xlabel`, `ylabel`, or `zlabel` functions

For example, this code creates 100 axes with titles in a tiled chart layout. It runs 11.9x faster than in the previous release:

```
function timingTitle  
tiledlayout(10,10);  
for n = 1:100  
    nexttile  
    title(n)  
end  
end
```

The approximate execution times are:

**R2021a:** 9.5 s

**R2021b:** 0.8 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the `timeit` function:

```
timeit(@timingTitle)
```

The performance gains increase with the number of axes, titles, and axis labels you are working with. For example, this table shows the improvements for looping over 10, 20, 50, and 100 axes with titles.

Number of Axes with Titles	Performance Gain
10	2.5x
20	4.2x
50	7.0x
100	11.9x

## Plot Interactions: Improved performance for rendering data tips and rotating scatter plots of large data sets

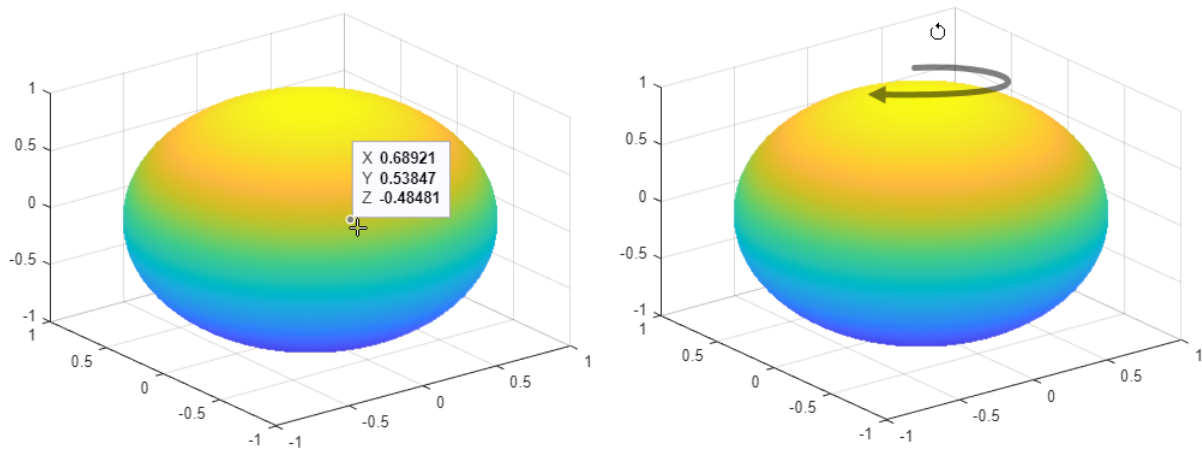
In figures created with the `uifigure` function and in MATLAB Online, interactions with scatter plots of large data sets have the following performance improvements:

- Data tip markers track the mouse motion more closely.
- 3-D scatter plots are more responsive to rotation gestures.

This improvement can be seen when the axes are created with either the `axes` or `uiaxes` function.

For example, on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz system, when you hover the cursor over the following sphere, the cursor changes to a crosshair more quickly, and the data tip markers track the cursor more closely. When you click and drag the cursor within the axes, the sphere rotates more quickly and tracks the cursor more closely.

```
f = uifigure;
ax = axes(f);
[X,Y,Z] = sphere(900);
scatter3(ax,X(:),Y(:),Z(:),[],Z(:),".")
```



## Plots in Apps: Improved performance for creating plots

The performance is improved for creating plots in apps or in figures created with the `uifigure` function. For example, create a figure and an axes object. Then plot 10,000 points. This code runs 14x faster in R2021b.

```
function timingPlot
% Create figure and axes
f = uifigure;
ax = axes(f);
drawnow;

% Create data vector
y = rand(1,10000);

% Plot the data
tic;
```

```
plot(ax,y);
toc;
end
```

The approximate execution times are:

**R2021a:** 0.14 s

**R2021b:** 0.01 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the `timingPlot` function.

## App Designer: Improved performance when opening Start Page and loading apps

When you use App Designer, these operations have improved performance:

- Opening the App Designer Start Page
- Opening an existing app

For example, opening App Designer by entering `appdesigner` in the Command Window loads the Start Page approximately 1.8x faster in R2021b than in R2021a the first time it is opened, and 2.1x faster in subsequent times. The approximate startup times are:

Release	First Startup	Subsequent Startups
<b>R2021a</b>	6.6 s	3.6 s
<b>R2021b</b>	3.6 s	1.7 s

Also, loading an app in App Designer shows improved performance. For example, after creating and saving a new blank app, opening the app in App Designer is about 1.3x faster in R2021b than in R2021a. The approximate loading times are:

**R2021a:** 1.98 s



**R2021b:** 1.56 s

The performance improvement is larger if you have additional toolboxes installed.

These operations were timed on a Windows 10, Intel Core i7-5600 CPU @ 2.60 GHz test system.

## App Designer: Improved performance when saving apps

Saving apps in App Designer after you edit an app function or property is faster in R2021b than in R2021a. The more lines of code in the app file, the greater the performance improvement becomes.

For example, on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system, in an app containing 10,000 lines of code, if you click **Property**  to create a new property and then click **Save**  to save the app, you can run the updated app sooner in R2021b than in R2021a.

The approximate save times are:

**R2021a:** 20 s


**R2021b:** 1.5 s

## Comparison Tool: Improved performance when loading and saving MLAPP files

When you use the Comparison Tool to compare and merge changes between app code in MLAPP files, these operations have improved performance:

- Loading the files into the Comparison Tool
- Saving the files after merging changes



For example, if you load two apps with 5000 lines of code into the Comparison Tool by clicking

**Compare**  in the App Designer toolstrip, you can compare and merge the files sooner in R2021b than in R2021a.

The approximate loading times are:

**R2021a:** 13 s

**R2021b:** 8 s

Also, if you use the Comparison Tool to merge changes between two apps with 5000 lines of code (for example, by clicking **Merge Mode** , merging the changes, and then clicking **Save Result** ) , you can compare the saved files sooner in R2021b than in R2021a.

The approximate save times are:

**R2021a:** 24 s

**R2021b:** 8 s

Both of these operations were timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system.

## uigridlayout Function: Improved performance when adding components spanning multiple columns with 'fit' width

The performance of parenting components to a grid layout manager created using the `uigridlayout` function has improved when the components span multiple columns with a `ColumnWidth` value of `'fit'`. The performance improvement gets better as the number of components spanning multiple columns and the number of columns spanned increases.

For example, this code creates a grid layout manager with 10 columns with a `ColumnWidth` value of `'fit'`, and then creates 50 labels that span all 10 columns. Performance in R2021b is about 4.7x faster than in R2021a.

```
function timingGridLayout
f = uifigure;
numrows = 50;
numcols = 10;
```

```
g = uigridlayout(f);
g.Scrollable = 'on';
g.RowHeight = repmat({'fit'},1,numrows);
g.ColumnWidth = repmat({'fit'},1,numcols);
drawnow

tic
for row = 1:numrows
    txt = ['This is a label in row ' num2str(row) ' that spans ' ...
          num2str(numcols) ' columns in the grid.'];
    lbl = uilabel(g,'Text',txt);
    lbl.Layout.Column = [1 numcols];
end
drawnow
toc
end
```

The approximate execution times are:

**R2021a:** 5.2 s

**R2021b:** 1.1 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system by calling the function `timingGridLayout`.

## **uigridlayout Function: Improved resizing performance when wrapping text in resizable columns**

The performance when you resize apps containing a grid layout manager created using the `uigridlayout` function has improved when both of these conditions hold:

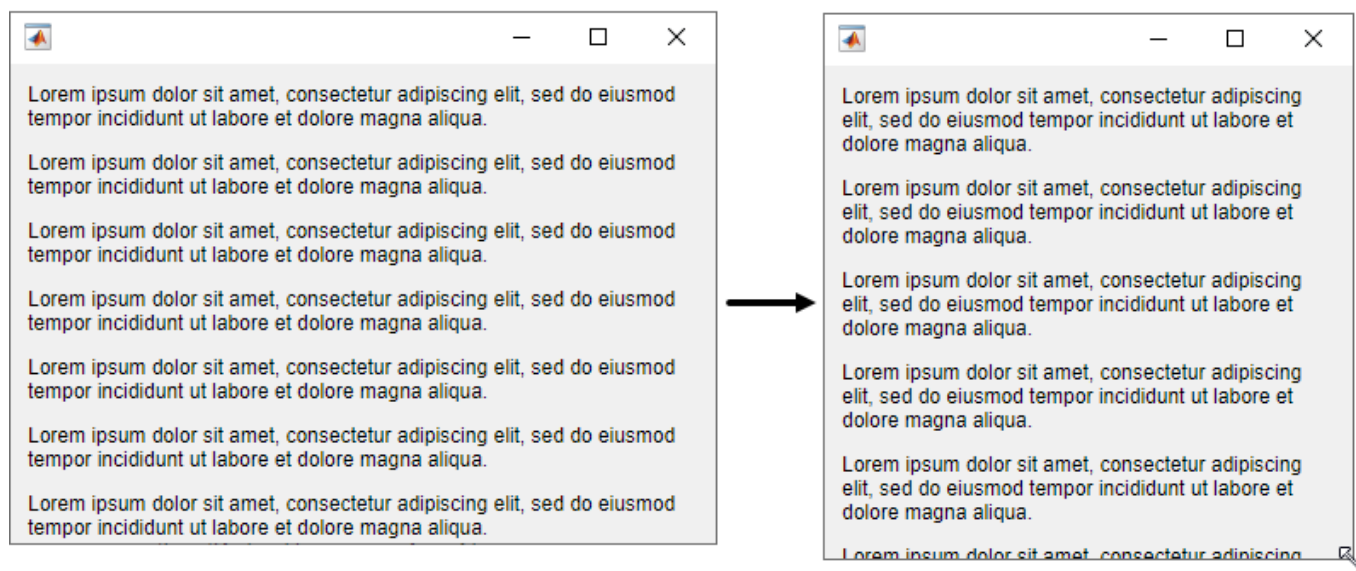
- The grid layout manager contains a component with a `WordWrap` value of `'on'`.
- The row and column containing the component with word wrap have a `RowHeight` of `'fit'` and a `ColumnWidth` that is resizable, such as `'1x'`.

For example, on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system, if you create 100 labels with wrapping text in a grid layout manager with fit height rows and resizable columns, and then resize the figure window by dragging the corner of the figure, the label text adjusts to fit the size of the figure almost immediately. In R2021a, there is a delay of about 2 seconds before the text adjusts.

```
f = uifigure;
g = uigridlayout(f);
g.ColumnWidth = {'1x'};
numrows = 100;
g.RowHeight = repmat({'fit'},numrows,1);

for row = 1:numrows
    c = uilabel(g);
    c.Text = ['Lorem ipsum dolor sit amet, consectetur adipiscing elit,' ...
             ' sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.'];
    c.WordWrap = 'on';
end
```





## Live Editor: Improved performance when saving live scripts or functions

Saving live scripts and live functions in the Live Editor is faster in R2021b than in R2021a. The improvement is most noticeable when you save live functions with more than 1000 lines of code and live scripts with fewer than 100 lines of code.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, saving an example live function containing 4000 lines of code takes approximately 2.05 seconds in R2021b. In R2021a, saving the same live function takes approximately 2.57 seconds.

## Data Processing Dialog Boxes: Improved resizing performance

The Basic Fitting UI, **Data Statistics UI**, Colormap Editor, and Linked Plot Data Sources dialog now use `uigriddlayout` to manage positions of UI components. This change results in a smoother experience when adjusting the size of these dialog boxes. For more information about the **Data Statistics UI**, see [Computing with Descriptive Statistics](#).

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, when you increase the size of the **Colormap Editor**, the size changes smoothly.

## Figure Interactions: Improved performance when using built-in axes interactions

Performance of figure interactions has been improved by coalescing built-in axes interactions so that there are significantly fewer interactions to process. These changes make interacting with a plot smoother and reduce the delay between an input and a response.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, while `panInteraction` mode is active, when you click and drag the cursor within the axes the figure pans more quickly and tracks the cursor more closely.

## **UI Figures: Improved performance when displaying axes toolbar**

The performance of the axes toolbar in UI figures has been improved to reduce the delay before the toolbar appears.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, when you pause the cursor on the axes, the axes toolbar appears more quickly.

## **UI Figures: Improved performance when interacting with linked axes**

Interacting with linked axes has improved performance when using figures created with the `uifigure` function or figures created in MATLAB Online™.

## Software Development Tools

### Projects: Collaborate using projects in MATLAB Online

Starting in R2021b, MATLAB Online provides support for basic projects workflows:

- Create an empty project and add files and folders.
- Clone a project from Git.
- Explore your project and run a dependency analysis.
- Create a project and manage your project files programmatically.

### Source Control: Work with files under Git in MATLAB Online

Starting in R2021b, MATLAB Online provides support for basic Git workflows:

- Cloning a remote Git repository
- Committing files to Git
- Pulling, pushing, and fetching files with Git

### Unit Testing Framework: Use the `TestCase` class template to create tests more quickly and accurately

You can now create a `TestCase` class, including basic test functionality, in MATLAB and MATLAB Online. To create a new test class, select **New > Test Class** on the **Home**, **Editor**, or **Live Editor** tabs.

Use the `TestCase` class template to create tests more conveniently. The template includes a `TestClassSetup` methods block, a `TestMethodSetup` methods block, and a `Test` methods block that defines a simple `Test` method. To customize your test class, add code to the file or remove unused code that is included by default. For more information about class-based tests, see [Author Class-Based Unit Tests in MATLAB](#).

### Unit Testing Framework: Run live-function-based tests interactively in MATLAB Online

Starting in R2021b, you can run live-function-based tests interactively in MATLAB Online. When you open an MLX file defining a function-based test in MATLAB Online, the toolstrip lets you run all tests in the file or just the current test.

To run tests and customize your test run interactively, use the **Run Tests** section in the **Live Editor** tab of the toolstrip. For more information, see [Run Tests in Editor](#).

### App Testing Framework: Perform press gestures on axes and UI axes with different selection types

The app testing framework now supports mouse selection types in `press` gestures that are performed on axes and UI axes. For example, create an axes with a plot and then test a double-click gesture at the point (3, 2).

```
f = uifigure;  
ax = axes(f);  
plot(ax,1:10)  
tc = matlab.uitest.TestCase.forInteractiveUse;  
tc.press(ax,[3 2], 'SelectionType', 'open')
```

## App Testing Framework: Perform drag gestures on axes and figures with different selection types

Starting in R2021b, the app testing framework supports drag gestures on UI figures. Additionally, when you test a drag gesture on an axes, UI axes, or UI figure, you can specify the mouse selection type. For example, create a figure and drag on it from the point (100, 200) to the point (200, 300) using a right-click gesture.

```
f = uifigure;  
tc = matlab.uitest.TestCase.forInteractiveUse;  
tc.drag(f,[100 200],[200 300], 'SelectionType', 'alt')
```

## App Testing Framework: Use any units of measurement in gestures at the center of components

Starting in R2021b, when you perform a gesture at the center of a component, the component or its parent containers can use any units of measurement. In previous releases, the framework does not support containers that use nonpixel units.

For example, create a figure and set its `Units` property to `'normalized'`. Then, create a panel in the figure and press at the center of the panel.

```
f = uifigure;  
f.Units = 'normalized';  
p = uipanel(f);  
tc = matlab.uitest.TestCase.forInteractiveUse;  
tc.press(p)
```

If you perform a gesture at the center of a component using a syntax that accepts location as an input (for instance, `press(testcase, comp, location)`), then the figure or parent containers can use only `'pixels'` as their units of measurement.

## Functionality being removed or changed

### Specifying diagnostic after name-value arguments in qualifications is not recommended

*Still runs*

Starting in R2021b, when you test for equality using the `verifyEqual`, `assumeEqual`, `assertEqual`, or `fatalAssertEqual` methods, specifying name-value arguments before the `diagnostic` input argument is not recommended. Place name-value arguments after all of the positional arguments instead. Although not recommended, you still can specify `diagnostic` after the name-value arguments when these arguments use the `name, value` syntax.

The reason for this change is that, starting in R2021a, MATLAB supports a new syntax for passing name-value arguments. In the new syntax, the name and value arguments are connected by an equals sign (`name=value`), and the name is not enclosed in quotes. To use the new syntax with qualification methods, specify positional arguments, including `diagnostic`, before the `name=value` arguments. If you specify `diagnostic` after `name=value` arguments, MATLAB produces an error.

This table shows an example of how you can update your code.

R2021a and Earlier	Starting in R2021b
<pre>testCase = matlab.unittest.TestCase.forInteractiveUse; verifyEqual(testCase,1.5,2, ...     "RelTol",0.1,"Difference must be within relative tolerance.",R assumeEqual(testCase,1,2, ...     "AbsTol",0.5,"Difference must be within absolute tolerance.",A</pre>	<pre>testCase; matlab.unittest.TestCase.forInteractiveUse verifyEqual(testCase,1.5,2, ...     "RelTol",0.1,"Difference must be within relative tolerance.",R assumeEqual(testCase,1,2, ...     "AbsTol",0.5,"Difference must be within absolute tolerance.",A</pre>

For more information, see `verifyEqual`.

### **matlab.unittest.TestSuite.fromProject ignores the files that do not define test procedures when creating a test suite**

*Behavior change*

Starting in R2021b, if your project includes files with the `Test` classification, `matlab.unittest.TestSuite.fromProject` ignores the files that do not define test procedures when you create a test suite. For example, if an abstract `TestCase` class definition file is labeled with the `Test` classification, `fromProject` ignores it. In previous releases, MATLAB produces an error if `fromProject` is called on a project that uses the `Test` classification for any files other than concrete test files. With this change, `fromProject` becomes consistent with the `matlab.unittest.TestSuite.fromFolder` method: both methods create a test suite from all the concrete test files and ignore any other files in the folder.

This behavior change also applies to the `testsuite`, `runtests`, and `runperf` functions when they operate on code organized into files and folders within a project.

### **Test suites created from projects cannot run without the Java Virtual Machine (JVM) software**

*Behavior change*

Starting in R2021b, if you start MATLAB without the Java® Virtual Machine (JVM™) software and create a suite from the test files in a project using `testsuite`, the function uses the `matlab.unittest.TestSuite.fromProject` method to create the suite. If you then try to run the test suite without the JVM software, MATLAB produces an error because the project cannot be opened without the JVM software. In previous releases, when MATLAB runs without the JVM software, `testsuite` uses `matlab.unittest.TestSuite.fromFolder` to create a suite from the test files in the project, and the testing framework runs the resulting test suite.

This behavior change also applies to the `runtests` and `runperf` functions when they operate on code organized into files and folders within a project.

## External Language Interfaces

### C++ interface: Support for C++ language features

The C++ interface supports these additional C++ language features.

#### Support for void\*\* parameters

MATLAB returns a `void*` argument for `void**` parameters. For more information, see `void**` Input Argument Types. For information about memory management of `void**` parameters, see `Pass Ownership of Memory to MATLAB`.

#### char [] parameters behave like char \* parameters

MATLAB supports `char []` parameters as either integer or character (string), the same as `char*` parameters. Likewise, the Unicode types `wchar_t []`, `char16_t []`, and `char32_t []` behave like `wchar_t*`, `char16_t*`, and `char32_t*`. For more information, see `C++ char* and char [] Types`.

#### Support for static data members

Public static and public const static data members are treated as read-only properties in MATLAB. You cannot modify the value of a C++ static data member in MATLAB. For more information, see `Static Data Members`.

You can use a public static data member (property) as the data type of an input argument or return type in a class constructor, method, or function. You also can use a static property or method to define the shape of an argument. For information about using static properties to define the shape, see `Use Property or Method as SHAPE`.

### C++ interface: Publisher options

The C++ interface supports these build configuration features.

#### Overwrite existing library definition files

Publishers can automatically overwrite existing library definition MLX files when calling `clibgen.generateLibraryDefinition`. Set the `OverwriteExistingDefinitionFiles` name-value argument to `true`. This option is useful when you create and modify the definition file for a library *libname*.

When you use this option, MATLAB deletes `definelibname.mlx` and `definelibname.m`, including any edits you made to the files.

#### Options for defining arguments

- By default, when a MATLAB input has fewer dimensions than the corresponding C++ argument, then MATLAB inserts singleton dimensions at the beginning of the `Shape` argument. For more information, see `Dimension Matching`. To insert singleton dimensions at the end, set the `'AddTrailingSingletons'` name-value argument to `true` in the `defineArgument` functions - `defineArgument (ConstructorDefinition)`, `defineArgument (FunctionDefinition)`, and `defineArgument (MethodDefinition)`.
- If a C++ function has a character array parameter used to return a C++ string, you can define the argument so that the function returns a null-terminated string. Use the

'NumElementsInBuffer' name-value argument in the `defineArgument` (`FunctionDefinition`) or `defineArgument` (`MethodDefinition`) functions. For an example, see the `getMessage` function in the Define String Argument table.

- You can transfer ownership of the memory of function or method parameters of double pointer type with scalar output to MATLAB using the `DeleteFcn` name-value argument. For more information, see `defineArgument` (`FunctionDefinition`) and `defineArgument` (`MethodDefinition`).
- MATLAB lets you control the lifetime management of objects created with a constructor by specifying the `ReleaseOnCall` name-value argument in the library definition file. For more information, see `defineArgument` (`ConstructorDefinition`) and Lifetime Management of C++ Objects in MATLAB.
- You can use a method or a property to define the shape of an argument. For more information, see Use Property or Method as `SHAPE`.
- You can transfer ownership to MATLAB of the memory of double pointer input arguments `void**` and `object**`. Use the 'DeleteFcn' name-value argument for `defineArgument`, as described in Pass Ownership of Memory to MATLAB.

## Java interface: Specify JRE path for MATLAB

You can run MATLAB with your system version of the Java Runtime Environment (JRE™). For information about Java versions compatible with MATLAB, see MATLAB Interfaces to Other Languages.

To set the JRE path in MATLAB, call `jenv`. You must restart MATLAB to use the updated path. This command sets the path for all future MATLAB sessions but does not change the path for other applications on your computer.

Alternatively, you can set the path from the operating system prompt. Call `matlab_jenv`, then start MATLAB.

## Java: Call into MATLAB from a Java program called by MATLAB

Java developers can use the `com.mathworks.engine.MatlabEngine` API `getCurrentMatlab` method to call back into MATLAB from Java. Incorporating this method in your application allows MATLAB users to call functionality from your Java program.

For information about developing these Java programs, see Call Back into MATLAB from Java.

## Python interface: Run Python commands and scripts from MATLAB

The `pyrun` and `pyrunfile` functions let you call Python commands and scripts from MATLAB. For more information, see Directly Call Python Functionality from MATLAB

## Python: Support for complex multidimensional arrays

MATLAB supports passing complex multidimensional array data to Python and from Python to MATLAB, for both in-process and out-of-process execution modes. For example, create a file `test.py` containing this code:

```
def returnData(data):  
    return data
```

To pass a complex MATLAB array to `returnData`, type:

```
mc = complex(magic(3));  
c = py.test.returnData(mc)
```

```
c =  
Python memoryview:
```

```
8.0000 + 0.0000i  1.0000 + 0.0000i  6.0000 + 0.0000i  
3.0000 + 0.0000i  5.0000 + 0.0000i  7.0000 + 0.0000i  
4.0000 + 0.0000i  9.0000 + 0.0000i  2.0000 + 0.0000i
```

Use `details` function to view the properties of the Python object.

Use `double` function to convert to a MATLAB array.

To convert the return value to a MATLAB array, type:

```
C = double(c)
```

```
C = 3x3 complex  
8.0000 + 0.0000i  1.0000 + 0.0000i  6.0000 + 0.0000i  
3.0000 + 0.0000i  5.0000 + 0.0000i  7.0000 + 0.0000i  
4.0000 + 0.0000i  9.0000 + 0.0000i  2.0000 + 0.0000i
```

For information about MATLAB to Python data type mapping, see [Pass Matrices and Multidimensional Arrays to Python](#).

## Python: Version 3.9 support

MATLAB now supports CPython 3.9, in addition to existing support for 2.7, 3.7, and 3.8. For more information, see [Versions of Python Compatible with MATLAB Products by Release](#)

## WSDL Web Services Documents: Apache CXF version 3.4.2 support

MATLAB supports Apache CXF version 3.4.2 for use with WSDL Web services. For more information, see [Set Up WSDL Tools](#).

## Version History

Download the latest version 3.4.2 release of the Apache CXF tool from <https://cxf.apache.org/download>.

## Perl 5.32.1: MATLAB support on Windows

As of R2021b, MATLAB on Windows ships with an updated version of Perl, version 5.32.1. See <https://www.perl.org> for a standard distribution of Perl, Perl source code, and information about using Perl.



## Version History

If you use the `perl` command on Windows platforms, see <https://www.perl.org> for information about using this version of the Perl programming language.

## Functionality being removed or changed

### **name=value syntax errors for calls to Python functions using `py.` prefix**

#### *Behavior change*

Starting in R2021b, MATLAB errors when you use `name=value` syntax for passing keyword arguments to Python functions using the `py.` prefix. In R2021a, MATLAB might silently give the wrong answer. Use `pyargs` to pass keyword arguments.

For example, the Python `print` function has a keyword argument `sep`. This Python statement sets the `sep` argument to a comma followed by a space:

```
print('comma','separated','values',sep=', ')
```

When you call this statement in MATLAB, MATLAB interprets `sep=', '` as a `name=value` argument:

```
py.print('comma','separated','values',sep=', ')
```

R2021a Behavior	R2021b Behavior	How to Update Your Code
<pre>py.print(...     'comma','separated','values',     sep=', ')</pre>	<pre>py.print(...     'comma','separated','values',     sep=', ')</pre>	<pre>py.print(...     'comma','separated','values',...     pyargs(sep=', '))</pre>
Silent wrong answer: <code>comma separated values sep ,</code>	Error: Error using <code>py.print</code> Using <code>name=value</code> format is not supported. Use <code>pyargs</code> to pass keyword arguments	<code>comma, separated, values</code>

### **`createSoapMessage`, `callSoapService`, and `parseSoapResponse` have been removed**

#### *Errors*

Consider using `matlab.wsdL.createWSDLClient` instead of the `createSoapMessage`, `callSoapService`, and `parseSoapResponse` functions to communicate with Web services using Simple Object Access Protocol (SOAP). There is no direct function replacement for the SOAP functions, but when you create a WSDL interface, you have access to the Web service functionality.

### **`createClassFromWsdL` has been removed**

#### *Errors*

The `matlab.wsdL.createWSDLClient` function replaces the `createClassFromWsdL` function to communicate with Web services from MATLAB using Web Services Description Language (WSDL). `matlab.wsdL.createWSDLClient` enables you to specify additional information needed to access the WSDL document. For more information, see `weboptions`.

To get started using `matlab.wsdL.createWSDLClient`, follow these steps.

- 1 Download supported versions of the Java JDK™ and Apache CXF programs. For more information, see `Set Up WSDL Tools`.

- 2 Set the paths to these programs, where `jdk` is the path to the JDK installation and `cxfrun` is the path to the CXF program.

```
matlab.wsdL.setWSDLToolPath('JDK',jdk,'CXF',cxfrun)
```

To update your code, replace calls to `createClassFromWsdL` with calls to `matlab.wsdL.createWSDLClient`. For example, for a Web service with this URL:

```
url = 'https://examplesite.com/samplewebservice';
```

replace this call to `createClassFromWsdL`:

```
createClassFromWsdL(strcat(url,'?WSDL'))
```

with:

```
matlab.wsdL.createWSDLClient(url)
```

---

**Note** `matlab.wsdL.createWSDLClient` does not support RPC-encoded WSDL documents.

---

## Hardware Support

### **Connect and Control Arduino board using the Arduino Explorer App**

The MATLAB Support Package for Arduino® Hardware now has an Arduino Explorer app.

Using this app, you can:

- Set up the Arduino board
- Connect to an Arduino board over USB, Bluetooth®, and WiFi
- Configure, read from, and write to Arduino pins
- Visualize data from Arduino pins
- Record and save data from Arduino pins to the MATLAB workspace
- Analyze the recorded data
- Generate equivalent MATLAB code

### **Read data from APDS9960 sensor connected to the Arduino hardware**

The MATLAB Support Package for Arduino Hardware enables you to read gesture, proximity, clear light and color (RGB) data from APDS9960 sensor connected to Arduino hardware.

### **Support for CAN shields on Raspberry Pi Hardware**

Use the MATLAB Support Package for Raspberry Pi™ Hardware to read and write CAN messages from the CAN network on the Raspberry Pi hardware.



# R2021a

---

**Version: 9.10**

**New Features**

**Bug Fixes**

**Version History**

## Environment

### Live Editor Controls: Create dynamic controls in live scripts by linking variables to drop-down items and slider values

When adding a drop-down list to a live script, you can populate the items in it using values stored in a variable. When adding a slider, you can specify the minimum, maximum, and step values using variables.

For example, create a live script and define the variable `lastnames` that contains a list of last names.

```
lastnames = ["Houston", "Vega", "Obrien", "Potter", "Rivera", "Hanson", "Fowler", "Tran", "Briggs"];
```

Run the live script to create `lastnames` and add it to the workspace. Then, go to the **Live Editor** tab, and in the **Code** section, select **Control > Drop Down**. In the **Items** section of the control configuration menu, select `lastnames` as the **Variable**.

The screenshot shows the configuration menu for a Drop Down control. It is divided into three sections: LABEL, ITEMS, and EXECUTION. In the LABEL section, the text 'Drop down' is entered. In the ITEMS section, the 'Item labels' list contains Houston, Vega, Obrien, and Potter. The 'Item values' list contains lastnames(1), lastnames(2), lastnames(3), and lastnames(4). The 'Variable' dropdown is set to 'lastnames'. In the EXECUTION section, the 'Run' dropdown is set to 'Current section'.

Close the configuration menu to return to the live script. The drop-down list now contains the last names defined in `lastnames`.

The screenshot shows a drop-down list with 'Houston' selected. The list contains the following items: Houston, Vega, Obrien, Potter, Rivera, Hanson, Fowler, Tran, and Briggs.

If you add, remove, or edit the values in `lastnames`, the items in the drop-down list update accordingly.

For more information, see [Add Interactive Controls to a Live Script](#).

## Live Editor Fonts: Change the name, style, size, and color of fonts programmatically using settings

You can change the name, style, size, and color of titles, headings, text, and code in the Live Editor using settings.

For example, this code changes the color and style of titles in the Live Editor:

```
s = settings;
s.matlab.fonts.editor.title.Style.PersonalValue = {'bold'};
s.matlab.fonts.editor.title.Color.PersonalValue = [0 0 255 1];
```

This code increases the size and changes the font of normal text in the Live Editor:

```
s = settings;
s.matlab.fonts.editor.normal.Size.PersonalValue = 20;
s.matlab.fonts.editor.normal.Name.PersonalValue = 'Calibri';
```

The Live Editor updates all open live scripts and live functions to show the selected fonts. When you create new live scripts or functions, they use the new fonts as well.

### Plot Random Data


This script plots a vector of random data and draws a horizontal line on the plot at the mean.

```
n = 50;
r = rand(n,1);
plot(r)

m = mean(r);
hold on
plot([0,n],[m,m])
hold off
title('Mean of Random Uniform Data')
```


For more information, see [matlab.fonts](#).




## Live Editor Display: Specify where to display output by default

You can change the default location for output in a new live script. Depending on this preference, new live scripts that you create display their output inline or on the right. To change the default, go to the **Home** tab, and in the **Environment** section, select  **Preferences**. Select **MATLAB > Editor / Debugger > Display**, and then select an option for the **Live Editor default view**:

- **Output on right** — Output displays to the right of the code. Each output displays next to the line that creates it. This option is ideal when writing code.
- **Output inline** — Output displays inline with the code. Each output displays underneath the line that creates it. This option is ideal for sharing.





## Live Editor Functions: Run live functions interactively using the Run button in MATLAB Online


In MATLAB Online, you can now run live functions interactively using the  **Run** button.

To run live functions that require input argument values or any other additional setup, configure the  **Run** button by clicking **Run**  and adding one or more commands. For more information about configuring the  **Run** button, see [Configure the Run Button for Functions](#).

## Live Editor Bookmarks: Navigate quickly between lines

To navigate quickly between lines, set bookmarks in your live scripts or live functions. Bookmarks are particularly useful in long files when you frequently need to move between sections.

To set a bookmark, put the cursor on the line that you want to set it on. Then, go to the **Live Editor** tab and, in the **Navigate** section, click  **Bookmark**. A bookmark icon  appears to the left of the line. To clear a bookmark, with the cursor on the line with the bookmark, click **Bookmark**  and select **Set/Clear**. You also can clear the bookmark by clicking the bookmark icon  to the left of the line.

To navigate to a bookmark, go to the **Live Editor** tab, and in the **Navigate** section, click **Bookmark** . Then, select **Previous** or **Next**.

For more information about navigating within files, see [Go To Location in File](#).

## Live Editor Animation Playback Controls: Interactive interface to control animations

Playback controls now appear within the figure window after an animation is done playing. These playback controls provide the ability to replay the animation and explore individual frames without having to re-run the entire live script. Animation playback controls are not supported for animations generated by the `movie` function.

## Live Editor Performance: Improved performance when saving large live scripts or functions


When saving large live scripts or functions, you can continue using the Live Editor sooner in R2021a than in R2020b. While you continue to use the Live Editor, MATLAB saves the file in the background. When MATLAB finishes saving the file, the asterisk (\*) next to the file name disappears, indicating that the file is saved.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, if you save a live script containing 35,000 lines of code and then click the title bar of another document open in the Live Editor, MATLAB immediately switches to the other open document. In R2020b, there is a noticeable delay before MATLAB switches to the other open document.



## Help Browser: View web documentation by default

Starting in R2021a, when you run MATLAB with an internet connection, the Help browser displays the web documentation by default. When you run MATLAB on a system without an internet connection, or if your internet connection becomes unavailable, the Help browser displays the installed documentation instead.

To change the default documentation location, on the **Home** tab, in the **Environment** section, click  **Preferences**. Select **MATLAB > Help** and change the **Documentation Location**. For more information, see Help Preferences.

## Documentation: View MATLAB documentation in French, Italian, and German

A subset of MATLAB documentation in French, Italian, and German is available on the web to licensed MATLAB users. Only a subset of the full documentation is available. For more information, see Translated Documentation.

## MATLAB Drive: Get the location of your MATLAB Drive root folder programmatically

You can get the location of your MATLAB Drive™ root folder programmatically, using the `matlabdrive` command, from your desktop or in other MATLAB environments such as MATLAB Online. For example, if you have MATLAB Drive Connector installed on your desktop system, MATLAB returns the location of your MATLAB Drive:

```
path = matlabdrive  
  
path =  
    'C:\Users\username\MATLAB Drive'
```

## Functionality being removed or changed

### PNG images in documentation are compressed

#### *Behavior change*

Starting in R2021a, all PNG images included in the documentation are compressed, reducing their stored size on disk. The compressed images should be visually identical to the original images.

## Language and Programming

### **Name=Value Syntax: Use name=value syntax for passing name-value arguments**

MATLAB supports a new syntax for passing name-value arguments. In the new syntax, the name and value arguments are connected by an equal sign, and the name is not enclosed in quotes.

**Name=value syntax:** `plot(x,y,LineWidth=2)`

**Comma-separated syntax:** `plot(x,y,"LineWidth",2)`

MATLAB continues to support the comma-separated name,value syntax. Existing functions and methods support both syntaxes, and the process for writing functions and methods with name-value arguments is unchanged.

Use the new syntax to help identify name-value arguments for functions and to distinguish names from values in lists of name-value arguments. There are some limitations on where and how the name=value syntax can be used:

- The recommended practice is to use only one syntax in any given function call. However, if you do mix name=value and name,value syntaxes in a single call, all name=value arguments must appear after the name,value arguments. For example, `plot(x,y,"Color","red",LineWidth=2)` is a valid combination, but `plot(x,y,Color="red","LineWidth",2)` errors.
- Similarly, name=value arguments must appear after all positional arguments. Calling `myFunction(name=value,posArgument)` errors.
- The name=value syntax can only be used directly in function calls. They cannot be wrapped in a cell array or additional parentheses. Calling `myFunction(a,(name=value))` errors.

### **Retrieving Display Format: format function can get and set display format**

The `format` function can now output the current Command Window display format. Calling `format` with an output variable returns a `DisplayFormatOptions` object that describes the current numeric and line spacing formats:

```
fmt = format
```

```
fmt =
```

```
DisplayFormatOptions with properties:
```

```
    NumericFormat: "shortE"  
    LineSpacing: "loose"
```

You can also use a `DisplayFormatOptions` object as an input to `format` to change the display format.



## eval function: Context checking to resolve identifiers

MATLAB now resolves identifiers like variable names in an `eval` statement using additional context. For instance, MATLAB recognizes when a call to `eval` uses a variable declared in a function.

### Version History

With the additional context, MATLAB resolves ambiguities differently than in previous releases. Some code now produces errors or warnings.

Code that warns starting in R2021a will error in a future release.

Example	Previous Result	R2021 Result	Notes
<pre>function myfun(pi)     eval('pi') end  &gt;&gt; myfun</pre>	<pre>ans =  3.1416</pre>	Errors	MATLAB resolves <code>pi</code> as a variable in the function workspace, so <code>myfun</code> now errors when <code>pi</code> is not defined.
<pre>function myfun     disp(pi)     eval('pi = 1'); end  &gt;&gt; myfun</pre>	<pre>3.1416  pi =  1</pre>	Same output but warns on assignment.	<p>MATLAB resolves <code>pi</code> as a variable in the function workspace.</p> <p>In a future release, <code>myfun</code> errors. This new behavior is consistent with processing the following code:</p> <pre>function myfun     disp(pi)     pi = 1; end</pre>
<pre>% assignLocal.m script local = 1;  % myfun.m file with local function: function myfun     local()     assignLocal     eval("local") end function local     disp("local fx") end  % function call: &gt;&gt; myfun</pre>	<pre>local fx  local =  local function:</pre>	Same output but warns on assignment.	<p>In a future release, MATLAB will give precedence as described in "Function Precedence Order". In this example, precedence goes to the local function. This new behavior is consistent with processing the following code:</p> <pre>function myfun     local()     assignLocal     local() end</pre>

## Functionality being removed or changed

### format with no arguments is not recommended

*Still runs*

The `format` command, by itself, resets the output display format to the default, which is the short, fixed-decimal format for floating-point notation and loose line spacing for all output lines.

```
format
```

For clearer code, explicitly specify the `default` style.

```
format default
```

### Defining classes and packages: Using `schema.m` will not be supported in a future release

*Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

### compose does not accept an invalid hexadecimal value, octal value, or trailing backslash

*Errors*

Previously, when the `formatSpec` input to `compose` contained an invalid hexadecimal value, octal value, or trailing backslash it would issue a warning and truncate the output at the point of the invalid value. Starting in R2021a, MATLAB will issue an error instead. With this change, `compose` will error for all invalid `formatSpec` inputs.

### Using `get` and `set` to access or change display format is not recommended

*Still runs*

Using `get` and `set` to programmatically access or change the numeric display format and the display line spacing is not recommended. Use `settings` instead. For example:

```
s = settings;
myformat = s.matlab.commandwindow.NumericFormat.ActiveValue

myformat =
    'short'

s.matlab.commandwindow.DisplayLineSpacing.TemporaryValue = 'compact';
myspacing = s.matlab.commandwindow.DisplayLineSpacing.ActiveValue

myspacing =
    'compact'
```

For more information, see `matlab.commandwindow` Settings.

### Using `feature('EightyColumns')` to access and change Command Window display width is not recommended

*Still runs*

Using the command `feature('EightyColumns')` or `feature('EightyColumns',value)` to programmatically determine or change whether the Command Window display width limit is enabled is not recommended. Use `settings` instead. For example:

```
s = settings;
s.matlab.commandwindow.UseEightyColumnDisplayWidth.TemporaryValue = 1;
limitwidth = s.matlab.commandwindow.UseEightyColumnDisplayWidth.ActiveValue
```

```
limitwidth =  
    logical  
         0
```

For more information, see [matlab.commandwindow Settings](#).

## Data Analysis

### Data Preprocessing Live Editor Tasks: Operate on multiple table variables and specify output format for table input

When you are working with data in a table or timetable, these Live Editor tasks now allow you to operate on multiple table variables at the same time:

- **Clean Missing Data**
- **Clean Outlier Data**
- **Smooth Data**
- **Remove Trends**
- **Find Local Extrema**
- **Find Change Points**

You can also choose which variable to display when visualizing the results.

In addition, tasks that modify variables provide new output options. You can return a table with all of the variables, or with only the variables that were modified. For tasks that return logical arrays, you can specify the size of the output. The size can match the size of the input table or a table containing only the variables that were used in the calculation.

### Clean Outlier Data Live Editor Task: Visualize results with a histogram

The **Clean Outlier Data** Live Editor task now offers histogram plots for most detection methods. The histogram can summarize the input data, the outliers, the cleaned data with the outliers filled, and the outlier detection thresholds and center value.

### fillmissing Function: Specify custom fill method

You can now specify a custom method for filling missing values when using the `fillmissing` function. Specify the custom method as a function handle.

### normalize Function: Normalize multiple data sets with same parameters

`normalize` can now return the centering and scaling parameter values used to perform the normalization. You can reuse these parameters to normalize subsequent data sets in the same manner. For example, you can normalize an array of data `A` and then normalize a second array `B` with the same parameters:

```
[Anorm,C,S] = normalize(A);
Bnorm = normalize(B, 'center',C, 'scale',S);
```

The new outputs `C` and `S` contain the centering and scaling parameter values, respectively. So that you can easily reuse them in a later normalization step, you can also now specify the `'center'` and `'scale'` normalization methods at the same time. These are the only two normalization methods that you can specify together.

To further support these changes, when method is 'center' or 'scale', the possible values of `methodtype` now include arrays and tables. While these `methodtype` values are intended to work with the new outputs C and S, you also can compute your own normalization parameters to specify.

## **groupcounts Function: Display percentages of group counts**

`groupcounts` now displays information about the percentage each group count represents.

- For table and timetable inputs, `groupcounts` automatically displays the percentages represented by each group count in the output table.
- For array inputs, `groupcounts` has a new third output argument to return the percentages represented by each group count.

## **Version History**

When `groupcounts` operates on data in a table or timetable, the output contains an additional table variable for the percentages. The percentages are in the range [0 100] and are included in the table variable `Percent`.

Any code that references specific table variables is unaffected. However, you might need to update code that depends on the number of variables in the output table.

## **ts2timetable Function: Convert timeseries objects to timetables**

To convert `timeseries` objects to timetables, use the `ts2timetable` function.

## **table and timetable Functions: Specify dimension names using the 'DimensionNames' name-value argument**

When you create tables and timetables, you can specify their dimension names by using the 'DimensionNames' name-value argument with these functions:

- `array2table`
- `array2timetable`
- `cell2table`
- `struct2table`
- `table`
- `timetable`

In previous releases, you could change the dimension names only by creating a table or timetable and then changing its `DimensionNames` property.

## **Functionality being removed or changed**

### **Table dimension names cannot match reserved names**

*Behavior change*



MATLAB raises an error if you assign a dimension name that matches one of these reserved names: 'Properties', 'RowNames', 'VariableNames', or ':'. In previous releases, MATLAB raised a warning and modified the dimension names so that they were different from the reserved names.

For example, if you create a table and then assign 'Properties' as a dimension name, the result is an error.

```
T = array2table(magic(3));  
T.Properties.DimensionNames = {'Rows', 'Properties'}
```

### **'SamplingRate' will be removed**

*Warns*

The 'SamplingRate' name-value argument will be removed in a future release. Use 'SampleRate' instead. The corresponding timetable property is also named `SampleRate`.

For backward compatibility, you still can specify 'SamplingRate' as the name of the name-value argument. However, the value is assigned to the `SampleRate` property.

This change in behavior affects the timetable functions shown in the list:

- `array2timetable`
- `retime`
- `synchronize`
- `table2timetable`
- `timetable`

## Data Import and Export

### XML Files: Read, write, and import XML files using `readtable`, `readtimetable`, and other functions

The `readtable`, `writetable`, `readtimetable`, `writetimetable`, and `detectImportOptions` functions now support reading and writing XML files. This list outlines the added capabilities of each function.

- `readtable` and `readtimetable` — Read XML data into MATLAB as a table or timetable. You can specify optional name-value arguments to control how `readtable` and `readtimetable` treat XML data. For example, specify `'ImportAttributes', false` to ignore attribute nodes.
- `writetable` and `writetimetable` — Write a table or timetable in MATLAB to an XML file. Specify optional name-value arguments to control how `writetable` and `writetimetable` treat XML data. For example, specify `'AttributeSuffix', '_att'` to specify that all table or timetable variables with the suffix `'_att'` should be written as attributes in the output XML file.
- `detectImportOptions` — The `detectImportOptions` function now returns an `XMLImportOptions` object when you call it on an XML file. Its behavior when you call it on other file types has not changed. Use the `XMLImportOptions` object with `readtable` to customize import options. For instance:
  - Import only a subset of data using the `SelectedVariableNames` property.
  - Specify the names and data types of the variables in the input file using the `VariableNames` and the `VariableTypes` properties.
  - Manage the import of specific nodes in the XML file using name-value arguments such as `'TableSelector'`, `'RowSelector'`, or `'VariableSelectors'`.

For more information, see `XMLImportOptions`.

### MATLAB API for Advanced XML Processing: Create, read, write, transform, and query XML

Use the MATLAB API for XML Processing (MAXP) to develop advanced applications that create, read, write, transform, and query XML documents. MAXP consists of these packages:

- `matlab.io.xml.dom` — Classes for creating, reading, and writing XML files and strings following the W3C DOM standard.
- `matlab.io.xml.transform` — Classes for transforming XML documents from one type to another following the XSLT 1.0 standard.
- `matlab.io.xml.xpath` — Classes for querying XML documents using XPath 1.0 expressions.

### XML Files: Register XML namespace prefixes for evaluating XPath expressions using `readtable`, `readstruct`, and other functions

Use the `RegisteredNamespaces` name-value argument to specify namespace prefixes that `readtable`, `readtimetable`, `readstruct`, `XMLImportOptions`, and `detectImportOptions` use when evaluating XPath expressions in an XML file. `RegisteredNamespaces` can be used when you also evaluate an XPath expression specified by a selector name-value argument, such as `StructSelector` for `readstruct`, or `VariableSelectors` for `readtable` and `readtimetable`.

The `readstruct` function automatically detects namespace prefixes to register for use in XPath evaluation, but you can also register new namespace prefixes using the `RegisteredNamespaces` name-value argument. You might register a new namespace prefix when an XML node has a namespace URL, but no declared namespace prefix in the XML file. In that case, you can specify `RegisteredNamespaces` as a string array containing a namespace prefix and the associated URL.

For example, evaluate an XPath expression on an XML file named `example.xml` which does not contain a namespace prefix declaration. Specify `'RegisteredNamespaces'` as `["myprefix", "https://www.mathworks.com"]` to assign the prefix `myprefix` to the URL `https://www.mathworks.com`.

```
data = readstruct("example.xml", "StructSelector", "/myprefix:Data", ...
    "RegisteredNamespaces", ["myprefix", "https://www.mathworks.com"])
```

In the resulting structure, the namespace prefix and URL will appear as attributes belonging to the element specified in the `StructSelector` name-value argument.

## Low-level file I/O functions and remote data: Perform read and write operations on remotely stored files

You can now use low-level file I/O functions, such as `fopen`, `fread`, and `fwrite`, to work with files stored in remote locations. Some supported remote locations include Amazon S3™ and Windows Azure® Blob Storage.

When reading from or writing to a remote location, you must specify the full path using a uniform resource locator (URL). For example, open a binary file from the Amazon S3 cloud.

```
fid = fopen("s3://bucketname/path_to_file/example.bin");
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## save and load functions and remote data: Save, load, and append data to remotely stored v7.3 MAT-files

You can now access v7.3 MAT-files stored in remote locations, such as Amazon S3 and Windows Azure Blob Storage, using the `save` and `load` functions.

When saving, loading, or appending data to a remote location, you must specify the full path using a uniform resource locator (URL). For example, load a MAT-file from the Amazon S3 cloud.

```
load("s3://bucketname/path_to_file/example.mat");
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## Reading Online Data: Read files over HTTP and HTTPS using readtable, audioread, and other reading functions

Read files from an internet URL by specifying `filename` as a string that contains the protocol type `'http://'` or `'https://'`. This lets you read data from their primary online sources.

This functionality is supported by these functions: `audioread`, `audioinfo`, `parquetread`, `parquetinfo`, `readtable`, `readtimetable`, `readvars`, `readstruct`, `readmatrix`, `readcell`, `readlines`, and `detectImportOptions`.

## **Parquet Data Format: Use categorical data in parquet data format**

Read, write, and analyze parquet data that contain the `categorical` data type.

This functionality is supported by these functions: `parquetread`, `parquetwrite`, `parquetinfo`, and `parquetDatastore`.

## **Datastores: Read all data from a datastore using parallel processing**

You can use parallel processing when reading all data from a datastore (requires Parallel Computing Toolbox). Parallel processing results in improved performance when reading data, especially with remote data.

## **Data Compression Functions: Improved functionality in zip/unzip and tar/untar**

On Windows, macOS, and Linux systems:

- `zip` can compress an individual file of any size.
- `zip` can compress any number of files in a single function call.
- `tar` can compress a group of files of any cumulative size.

Additionally, on Windows systems, `unzip` and `untar` replace invalid characters with underscores if they occur in entry path names of the original file.

## **imfinfo function: Get information about all Adobe Digital Negative (DNG) file tags**

The `imfinfo` function returns information on all DNG file tags as individual named fields in the output structure. For a complete list of DNG file tags, see Chapter 4 of the Adobe Digital Negative (DNG) Specification.

## **jsonencode: Add indentation to JSON text**

Use the `jsonencode` 'PrettyPrint' option to display JSON text with an indentation of two spaces.

```
s.Width = 800;  
s.Height = 600;  
s.Title = 'View from the 15th Floor';  
s.Animated = false;  
s.IDs = [116, 943, 234, 38793];  
jsonencode(s, 'PrettyPrint', true)
```

```
ans =  
{  
    "Width": 800,  
    "Height": 600,
```

```

    "Title": "View from the 15th Floor",
    "Animated": false,
    "IDs": [
        116,
        943,
        234,
        38793
    ]
}

```

## Functionality being removed or changed

### The `H5I.get_name` function only accepts named HDF5 datatypes as input arguments.

#### *Behavior change*

Starting in R2020a, the `H5I.get_name` function only accepts committed (previously called named) HDF5 datatypes as input arguments, and will error if you pass other datatypes as input. In releases R2019b and earlier, `H5I.get_name` does not error if you pass other datatypes as input.

To verify that the input is a committed HDF5 datatype, call the `H5T.committed` function on it. The `H5T.committed` function returns a value of 1 if the input is a committed HDF5 datatype, and a value of 0 if it is not.

### `imfinfo` now returns Adobe DNG tags belonging to versions 1.2 through 1.5 in individual named fields in the output structure

#### *Behavior change*

When you call the `imfinfo` function on an Adobe DNG file, it now returns tags belonging to versions 1.2 through 1.5 as individual named fields in the output structure. Previously, tags belonging to these versions were stored in the 'UnknownTags' field of the output structure. For a complete list of DNG file tags, see Chapter 4 of the Adobe Digital Negative (DNG) Specification.

### `imread` reads the first frame in a GIF file by default

#### *Behavior change*

Starting in R2021a, when you read a GIF file without specifying additional arguments, the `imread` function reads the first frame by default. Previously, `imread` read all the frames in the file by default.

To read all the frames in the order that they appear in the GIF file, specify the value of the 'Frames' name-value argument as 'all'.

### `serial` function will be removed

#### *Still runs*

`serial` and its object properties will be removed. Previously, `serial` and its object properties were not recommended. Use `serialport` and its properties instead.

This example shows how to connect to a serial port device using the recommended functionality.

Functionality	Use This Instead
<pre> s = serial("COM1"); s.BaudRate = 115200; fopen(s) </pre>	<pre> s = serialport("COM1",115200); </pre>

See [Transition Your Code to serialport Interface](#) for more information about using the recommended functionality.

## Mathematics

### Graph Algorithms: Compute all paths, all cycles, and cycle basis

graph and digraph objects have new functions to compute paths and cycles:

- `allpaths` — Compute all paths between two nodes in a graph or digraph object.
- `allcycles` — Compute all cycles in a graph or digraph object.
- `cyclebasis` — Compute the fundamental cycle basis of a graph object.
- `hascycles` — Determine whether a graph or digraph object contains cycles.

### griddedInterpolant Object: Use multivalued interpolation to interpolate multiple data sets simultaneously

`griddedInterpolant` can now interpolate multiple data sets on the same grid at the same query points. For example, if you specify a 2-D grid, a 3-D array of values at the grid points, and a 2-D collection of query points, then `griddedInterpolant` returns the interpolated values at the query points for each 2-D page in the 3-D array of values.

Previously, this functionality was available in `interp1` for 1-D interpolation, but this improvement to `griddedInterpolant` adds support for N-D multivalued interpolation.

### eig Function: Improved algorithm for skew-Hermitian matrices

`eig` now has an improved algorithm for input matrices that are skew-Hermitian. With the function call `[V,D] = eig(A)`, where `A` is skew-Hermitian, `eig` now guarantees that the matrix of eigenvectors `V` is unitary and the diagonal matrix of eigenvalues `D` is purely imaginary.

### cdf2rdf Function: Improved algorithm for all inputs

`cdf2rdf` has an improved algorithm for all input matrices that reduces floating-point round-off errors in the calculation.

### Functionality being removed or changed

#### Line Continuation: Ellipsis following operator treated as a space

*Behavior change*

Previously, if an ellipsis followed an operator inside of a matrix or cell array, it caused the operator to be treated as unary. Ellipses will now be treated as a space in all cases.

Old Behavior	New Behavior
<pre>x = [1 -... 2] x =     1 -2</pre> <p>Previously, this code was equivalent to the expression, <code>x = [1 -2]</code>.</p>	<pre>x = [1 -... 2] x =     -1</pre> <p>Now, the ellipsis will be treated as a space so this code is equivalent to the expression, <code>x = [1 -2]</code>.</p>



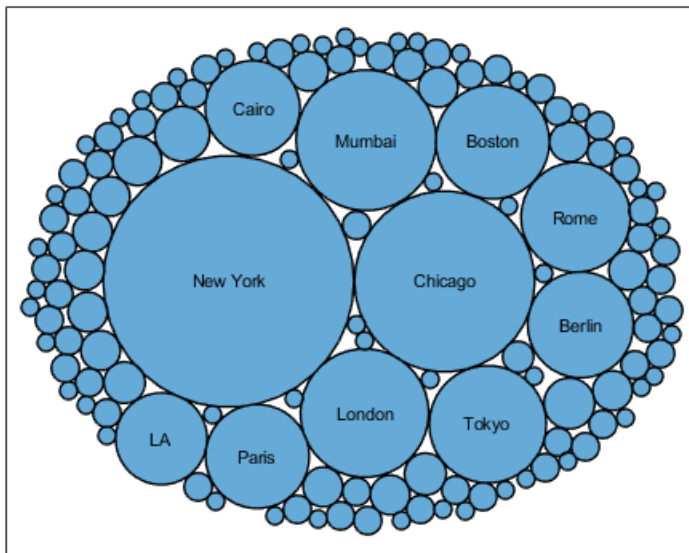
## Graphics

### Create Plot Live Editor Task: Create plots interactively and generate code

The Create Plot Live Editor Task makes generating and exploring visualizations for data simple and interactive. With this task you can select the data you wish to visualize and choose the plot type that best represents that data. Alternatively, you can explore the visualizations available in MATLAB to find the desired plot type and add your data. This task creates labels for the visualization based on the data and can be used to add or adjust the optional parameters of the visualization.

### bubblecloud Function: Visualize part-to-whole relationships

Use the `bubblecloud` function to illustrate the relationship between elements in your data set and the set as a whole. For example, you can visualize data collected from different cities, and represent each city as a bubble whose size is proportional to the value for that city.



### tilelayout Function: Control the tile indexing scheme

Control whether the tile indices increase across the rows or down the columns of a layout by setting the `TileIndexing` property of a `TiledChartLayout` object. Select one of the following options:

- `'rowmajor'` — Increment the tile indices across the first row before moving to the next row. This is the default behavior.
- `'columnmajor'` — Increment the indices down the first column before moving to the next column. This indexing scheme is the same as linear indexing for arrays.

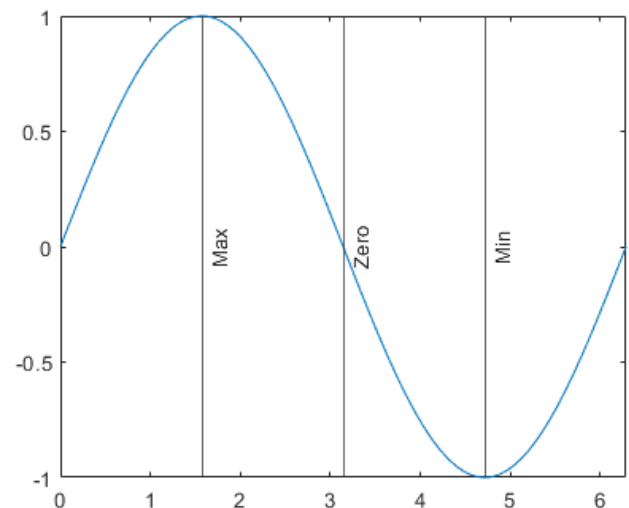
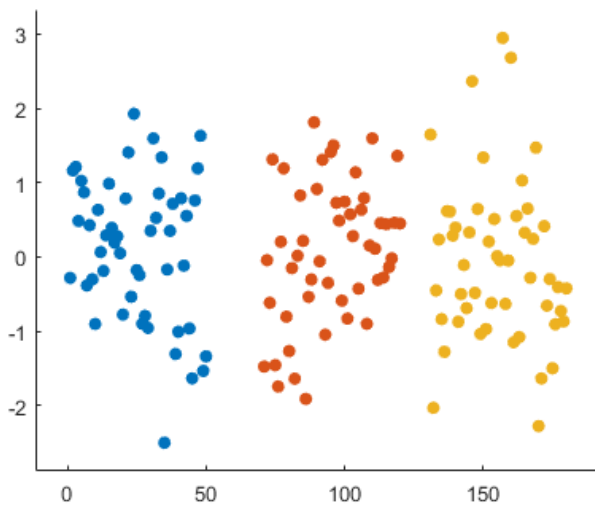
The `nexttile` function populates tiles according to the indexing scheme. If you change the tile indexing of a populated layout, the tile positions change to match the new scheme.

## PolarAxes Objects: Use the CurrentPoint property or call ginput to get the cursor location within polar axes

Query the `CurrentPoint` property of a `PolarAxes` object to get the location of the last click within the axes. The `ginput` function also supports querying coordinates of polar axes.

## Scatter Plots and Constant Lines: Create multiple scatter plots or constant lines at once

- The `scatter`, `polarscatter`, and `swarmchart` functions now accept the same combinations of matrices and vectors as the `plot` function does. As a result, you can visualize multiple data sets at once rather using the `hold` function between plotting commands.
- The `xline` and `yline` functions now accept vectors of values for creating multiple vertical or horizontal reference lines. You can also specify separate labels for each line using a string array or a cell array.



## Axis Limits: Define LimitsChangedFcn callback that executes when the limits of an axis change

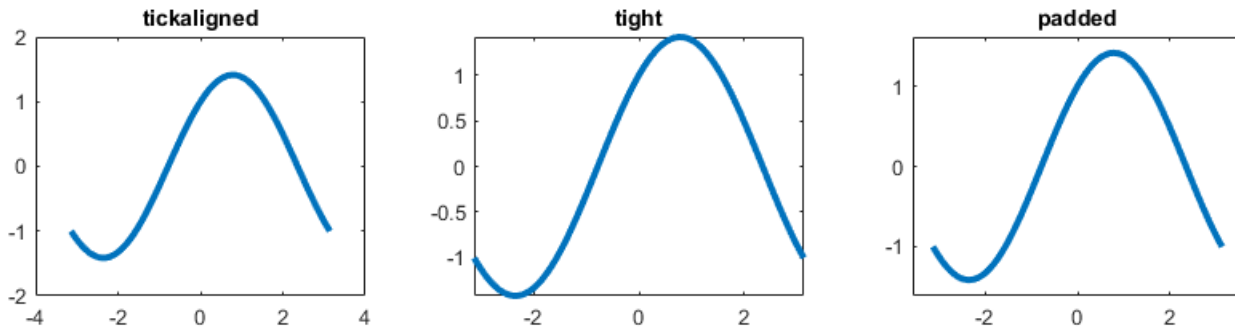
Define the `LimitsChangedFcn` callback function on any type of ruler object such as a numeric ruler. The callback function executes when the limits of the corresponding axis change. For example, you can define the callback in an app to update another aspect of the UI when the user pans within the axes.

## Axis Limits: Control axis limits

Control the axis limits for your plots by setting the `XLimitMethod`, `YLimitMethod`, or `ZLimitMethod` on the axes. Select one of the following property values:

- `'tickaligned'` — Align the edges of the axes box with the tick marks that are closest to your data without excluding any data. This is the default option.

- 'tight' — Fit the axes box tightly around the data by setting the axis limits equal to the range of the data.
- 'padded' — Fit the axes box around the data with a thin margin of padding on each side. The width of the margin is approximately 7% of your data range.



## exportgraphics and copygraphics Functions: Specify RGB, CMYK, or grayscale output

Choose a color space when exporting or copying graphics. Specify the `Colorspace` name-value argument when you call the `exportgraphics` or `copygraphics` functions. Select one of the following options:

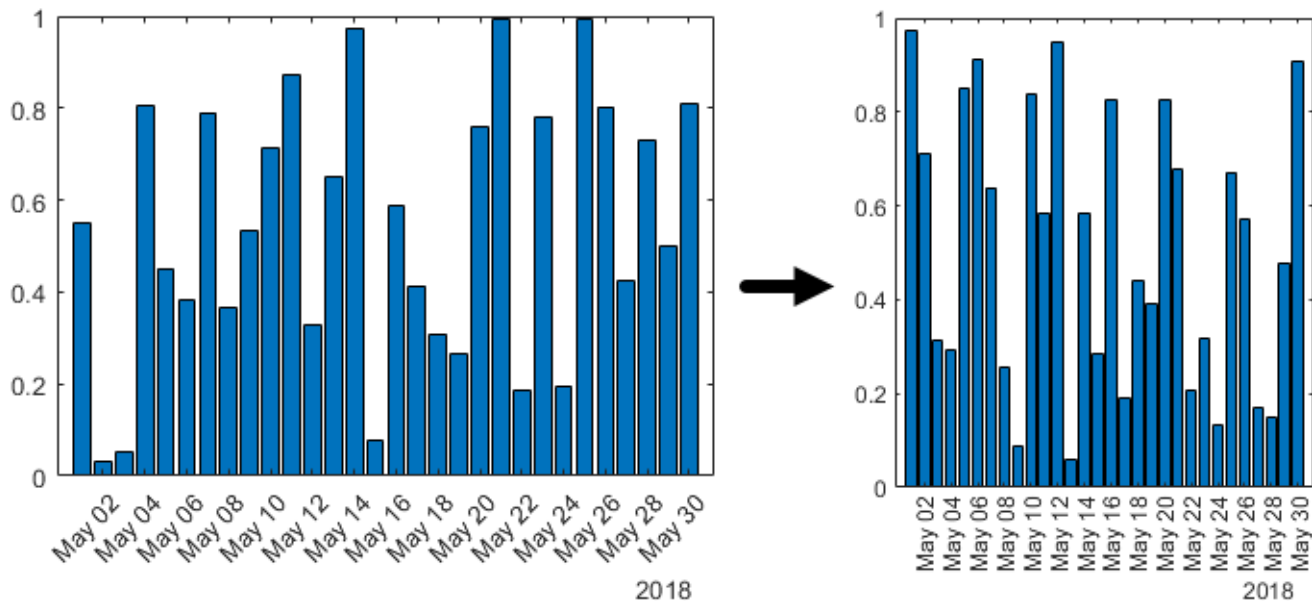
- 'rgb' — Capture truecolor RGB content. This is the default color space.
- 'gray' — Convert the content to grayscale.
- 'cmyk' (`exportgraphics` only) — Convert the content to cyan, magenta, yellow, and black (CMYK) before exporting the content as an EPS file.

## colororder Function: Control colors in stacked plots

The `colororder` function now supports charts created with the `stackedplot` function.

## Tick Labels: Automatically rotate tick labels

When you manually specify the ticks or the tick labels for a chart, the tick labels automatically rotate to give the best possible presentation given the size of the figure and the number of the tick labels.



## patch and errorbar Functions: Expanded data type support

The patch and errorbar functions now support more data types:

- The patch function accepts numeric, datetime, duration, and categorical values for the x-, y-, and z-coordinates.
- The errorbar function accepts numeric, datetime, duration, and categorical values for the x- and y-coordinates. It also accepts numeric and duration values for the error bar lengths above, below, and on either side of the data points.

## Geographic Plots: Access basemaps using additional proxy server authentication types

You can now access basemaps for geographic axes and charts using additional proxy server authentication types.

- On Windows, you can use Basic, Digest, NTLM, Negotiate (SPNEGO), and Kerberos authentication.
- On Linux and macOS, you can use Basic, Digest, and NTLM authentication.

Prior to R2021a, geographic axes and charts supported only types without authentication or with Basic authentication. For more information about specifying proxy server settings, see [Use MATLAB Web Preferences For Proxy Server Settings](#).

## Functionality being removed or changed

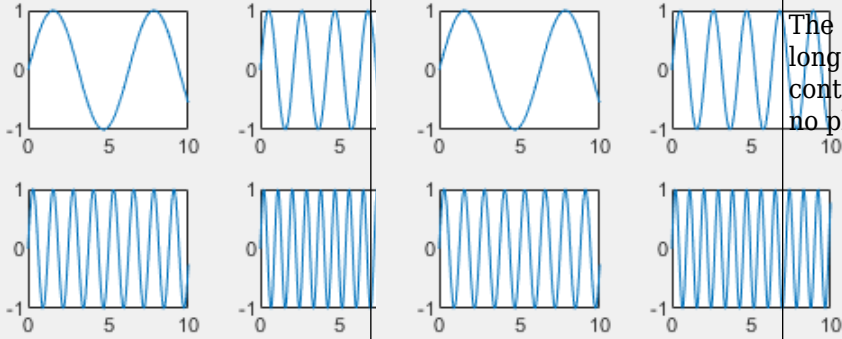
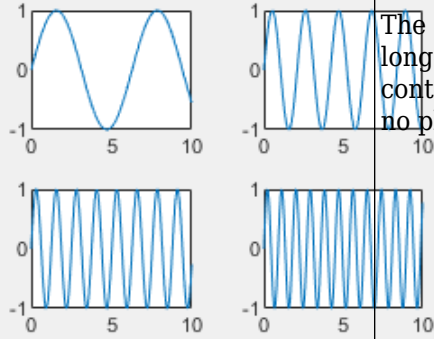
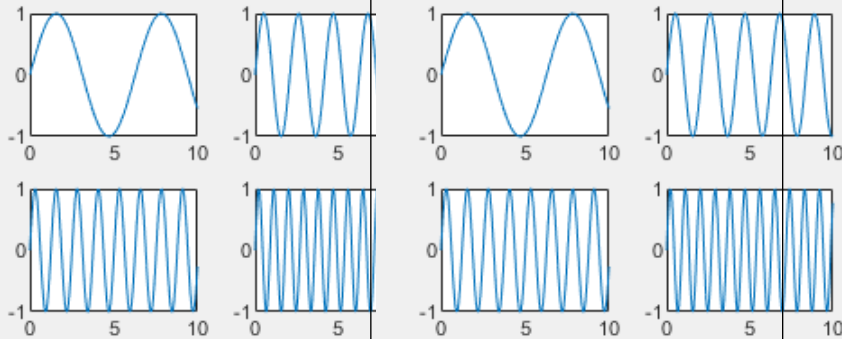
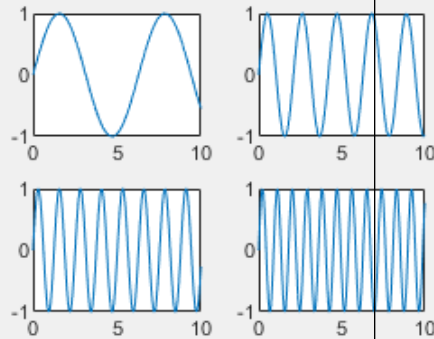
### Tile spacing and padding options for tiled chart layouts have changed

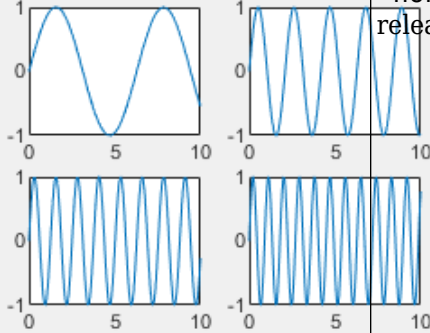
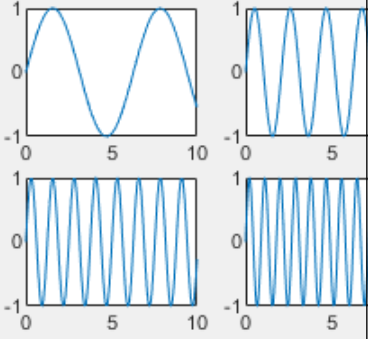
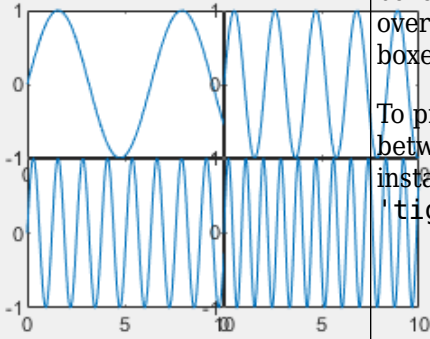
*Behavior change*

When you create a tiled chart layout, some of the `TileSpacing` and `Padding` properties provide a different result or have new names.

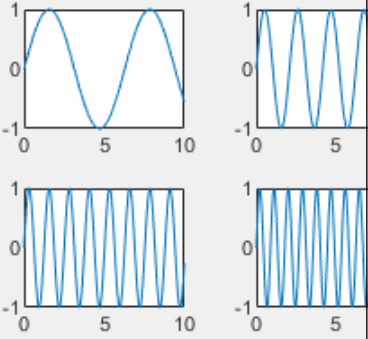
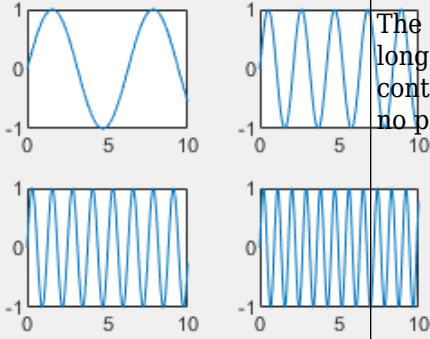
The new `TileSpacing` options are `'loose'`, `'compact'`, `'tight'`, and `'none'`. The new `Padding` options are `'loose'`, `'compact'`, and `'tight'`. The following tables describe how the previous options relate to the new options.

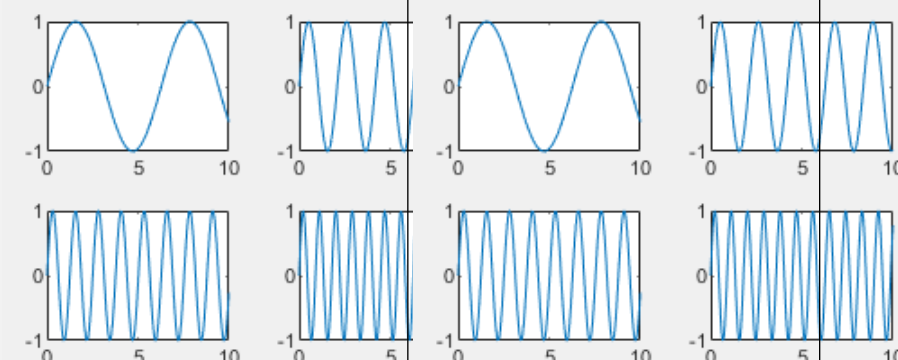
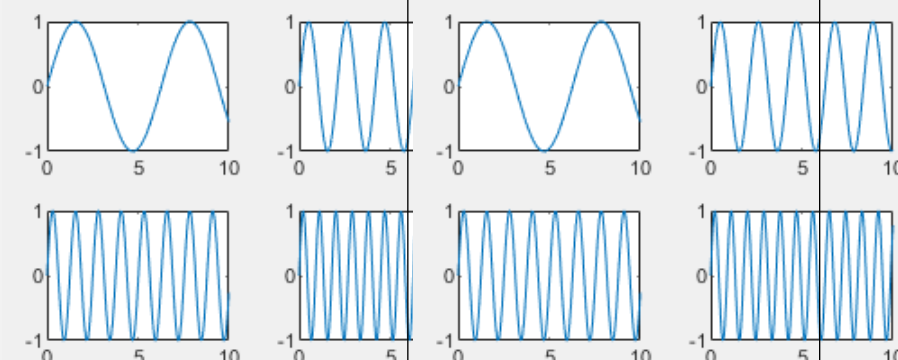
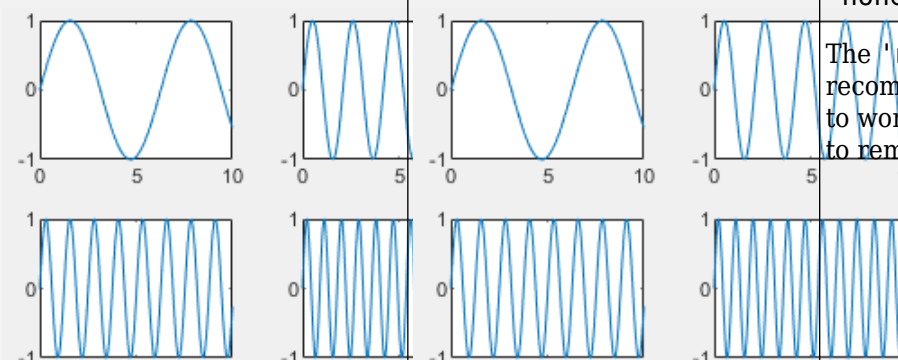
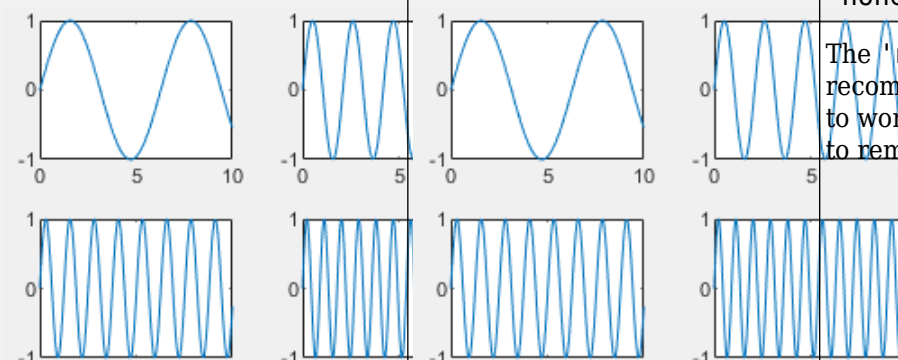
### TileSpacing Changes

Previous TileSpacing Option	R2021a TileSpacing Option	How to Update Your Code
<p data-bbox="248 600 370 625">'normal'</p> 	<p data-bbox="662 600 768 625">'loose'</p> 	<p data-bbox="1068 600 1453 657">Consider changing instances of 'normal' to 'loose'.</p> <p data-bbox="1068 688 1461 814">The 'normal' option is no longer recommended, but it continues to work, and there are no plans to remove it.</p>
<p data-bbox="248 1045 386 1071">'compact'</p> 	<p data-bbox="662 1045 800 1071">'compact'</p> 	<p data-bbox="1068 1045 1312 1071">No changes needed.</p>

Previous TileSpacing Option	R2021a TileSpacing Option	How to Update Your Code
<p>Not Applicable</p>	<p>'tight'</p> 	<p>'tight' is a new option. It provides the same result as 'none' does in previous releases.</p>
<p>'none'</p> 	<p>'none'</p> 	<p>The 'none' option removes all the space between adjacent plot boxes, and the tick labels overlap with neighboring plot boxes.</p> <p>To preserve the spacing between the plot boxes, change instances of 'none' to 'tight'.</p>

**Padding Changes**

Previous Padding Option	R2021a Padding Option	How to Update Your Code
<p>'normal'</p> 	<p>'loose'</p> 	<p>Consider changing instances of 'normal' to 'loose'.</p> <p>The 'normal' option is no longer recommended, but it continues to work, and there are no plans to remove it.</p>

Previous Padding Option	R2021a Padding Option	How to Update Your Code
'compact' 	'compact' 	No changes needed.
'none' 	'tight' 	Consider changing instances of 'none' to 'tight'. The 'none' option is no longer recommended, but it continues to work, and there are no plans to remove it.

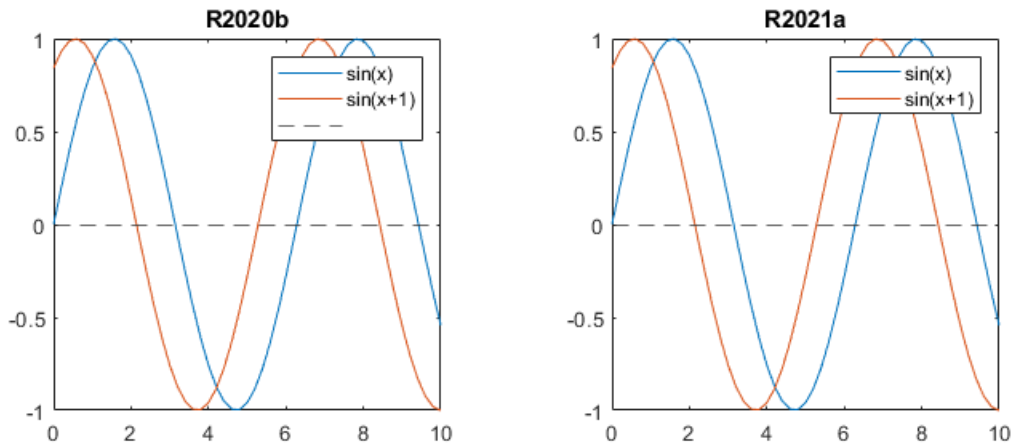
### Passing an empty label to the legend function omits the entry from the legend

#### *Behavior change*

When you call the `legend` function and specify a label as an empty character vector, an empty string, or an empty element in a cell array or string array, the corresponding entry is omitted from the legend. In R2020b and earlier releases, the entry appears in the legend without a label.

For example, this code plots two sine waves and a reference line at  $y=0$ . Then it creates a legend with three labels, where the last label is empty. In R2020b, the third line appears in the legend without a label. In R2021a, the third line is omitted from the legend.

```
x = 0:0.2:10;
plot(x,sin(x),x,sin(x+1));
hold on
yline(0,'--')
legend('sin(x)', 'sin(x+1)', '')
```



To keep an entry in the legend without a label, include a space character in the label. For example, to update the preceding code, specify the last label as a character vector containing a space (' ').

```
legend('sin(x)', 'sin(x+1)', ' ')
```

Alternatively, if you do not want to display a space character, you can pass the individual line objects to the legend function with an array of labels. To get the individual line objects, call each plotting function with an output argument.

```
x = 0:0.2:10;
p = plot(x, sin(x), x, sin(x+1));
hold on
line0 = yline(0, '--');
legend([p(1) p(2) line0], {'sin(x)', 'sin(x+1)', ' '});
```

### The XData, YData, and ZData properties on Patch objects created with the fill and fill3 functions return values of the original data type

#### Behavior change

The XData, YData, and ZData properties on a Patch object created by the fill or fill3 functions return the coordinates using the original input data type, rather than returning them as double values.

In previous releases, datetime, duration, and categorical coordinates are converted to double values when they are stored in the XData, YData, and ZData properties.

For example, this code creates a filled polygon with datetime x-coordinates. Then it calculates x2 using the values stored in the XData property. In R2020b, h.XData and x2 are double arrays. In R2021a, h.XData and x2 are datetime arrays.

```
x = datetime('01-Jan-2018') + days([0 1 1 0]);
y = [0 0 1 1];
h = fill(x, y, 'red');
x2 = h.XData + 1;
```

To preserve the double values in your code, get the double values from the Vertices property of the Patch object. The x-, y-, and z-coordinates are stored as double values in the first, second, and third columns of the Vertices array.

```
x2 = h.Vertices(:,1) + 1;
```



Alternatively, use the `ruler2num` function. Pass the coordinate values and the corresponding axis ruler to the `ruler2num` function.

```
ax = gca;
x2 = ruler2num(h.XData,ax.XAxis) + 1;
```

### **Graphics objects and UI components sized using 'points', 'inches', and 'centimeters' units will increase in size on macOS platforms**

*Behavior change in future release*

In a future release, graphics and UI objects that have `Units` or `FontUnits` properties set to 'points', 'inches', or 'centimeters' will use a conversion value of 1 pixel = 1/96th inch on macOS platforms. The current conversion value is 1 pixel = 1/72nd inch. As a result, these objects and text elements will display 1.33 times larger than their previous size. This change will provide a more readable default font size and will ensure a consistent object size across Windows and macOS platforms.

The following objects use a unit value of 'pixels' by default and will not be affected by this change:

- UI components in App Designer or in apps created with the `uifigure` function
- UI components in apps created in GUIDE and migrated to App Designer using the GUIDE to App Designer Migration Tool for MATLAB
- Axes objects created using the `uiaxes` function

For the most control when sizing and laying out your graphics objects and UI components, use a value of 'pixels' for `Units` and `FontUnits` properties. To maintain object sizes in current and future releases, make these updates in your code:

- Objects with `Units` or `FontUnits` set to 'points' — Update the value of the property from 'points' to 'pixels'.
- Objects with `Units` or `FontUnits` set to 'inches' — Update the value of the property from 'inches' to 'pixels' and multiply all `Position` values by 72.
- Objects with `Units` or `FontUnits` set to 'centimeters' — Update the value of the property from 'centimeters' to 'pixels' and multiply all `Position` values by 72/2.54.

For example, this code creates a push button in a figure window, with its position specified in inches:

```
uicontrol('Units','inches','Position',[0.6 0.1 1.75 0.5]);
```

In a future release, the push button created by this code will display 1.33 times larger on macOS platforms. To maintain the size and position of the push button in current and future releases, update the code to:

```
uicontrol('Units','pixels','Position',[50 10 126 36]);
```

### **Align Distribute Tool will be removed in a future release**

*Warns*

The Align Distribute Tool will be removed in a future release.

To control the arrangement of multiple plots in a figure, create a tiled chart layout using the `tiledlayout` function instead.

To align or distribute graphics objects within a figure, select **Tools > Align** or **Tools > Distribute** from the figure toolbar instead.

## App Building

### **uihyperlink Function: Add and configure clickable links in apps and on the App Designer canvas**

To create a link, call the `uihyperlink` function or, in App Designer, drag a hyperlink UI component from the **Component Library** onto the canvas.

Hyperlink UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

### **uitree Function: Add and configure check box trees in apps and on the App Designer canvas**

A check box tree is a styled with check boxes to the left of every item. You can now create check box trees in apps and on the App Designer canvas. Check box trees allow for easier selection of multiple tree nodes.

In apps created programmatically with the `uifigure` function, create a check box tree using the `uitree` function by specifying the style `'checkbox'`.

In App Designer, create a check box tree by dragging it from the **Component Library** onto the canvas.

### **Interpreter Property: Style text and display equations in labels with HTML and LaTeX markup**

Use the Interpreter property on Label objects (created with the `uicontrol` function) to enable markup in the label text. To add HTML markup, set the Interpreter property to `'html'`. To add LaTeX markup, set the Interpreter property to `'latex'`.

For more information, see `Label` Properties.

### **WindowState Property: Create UI figures that remain in the foreground**

To keep a specific UI figure window in front of other windows, set the `WindowState` property to `'alwaysontop'`. Unlike modal figures, UI figure windows with this property setting do not restrict keyboard and mouse interactions.

The `'alwaysontop'` property value is available only in figures created using the `uifigure` function.

For more information, see `UI Figure` Properties.

### **scroll Function: Scroll to a location within a table UI component programmatically**

Use the `scroll` function to scroll within a table UI component programmatically. Specify the scroll location as the top, bottom, left, or right side of the table, or as a specific row, column, or table cell.

## UI Component Accessibility: Select ListBox items, Table cells, ColorPicker colors, and DatePicker menus using the keyboard

You can now use keyboard shortcuts to change the focus and make selections in various UI components. These shortcuts are supported for components in figures created using the `uifigure` function.

Component	Action	Keyboard Shortcut
ListBox with Multiselect property set to 'off'	Toggle list box selection.	<b>Space</b>
ListBox with Multiselect property set to 'on'	Select all items.	<b>Ctrl+A</b>
	Move focus to previous item and toggle selection.	<b>Shift+Up</b>
	Move focus to next item and toggle selection.	<b>Shift+Down</b>
	Move focus to previous item without toggling selection.	<b>Ctrl+Up</b>
	Move focus to next item without toggling selection.	<b>Ctrl+Down</b>
	Toggle selection of item currently in focus.	<b>Ctrl+Space</b>
Table	Select table cell in top-left corner.	<b>Ctrl+Home</b>
	Select table cell in bottom-right corner.	<b>Ctrl+End</b>
ColorPicker gradient selector	Move gradient selector.	Arrow keys
ColorPicker hue slider	Move hue slider.	<b>Up and Down</b>
DatePicker	Cycle between drop down menus, buttons, and calendar.	<b>Tab</b>

## App Designer: Use custom UI components in App Designer

You can now configure custom UI component classes to appear in the App Designer **Component Library** and to be used interactively in **Design View**.

To configure a custom UI component class for use in App Designer, follow these steps:

- Define your own UI component class by creating a subclass of the `matlab.ui.componentcontainer.ComponentContainer` base class.
- Call the `appdesigner.customcomponent.configureMetadata` function and specify the path to the component class `.m` file.
- Use the resulting dialog box to configure the metadata associated with the component, including the component name, icon, author, and category.

This creates a `resources` folder with the App Designer metadata. To view the component in the App Designer **Component Library**, add the folder containing the component class file and the generated `resources` folder to the MATLAB path.

To share your custom UI component for others to use in App Designer, share both the component class file and the `resources` folder.

For more information, see [Configure Custom UI Components for App Designer](#).

## App Designer: Zoom and pan in the canvas, and zoom in the Code View editor

To zoom in or out in the App Designer canvas and in the **Code View** editor, hold **Ctrl** and move the scroll wheel, or press **Ctrl+Plus** and **Ctrl+Minus**. To return to the default scale, press **Ctrl+Alt+0**. Zooming does not affect the **Component Library**, **Component Browser**, or **Code Browser**.

To pan in the App Designer canvas, use one of the following:

- **Click** and drag with the middle mouse button.
- Hold **Space** while clicking and dragging with the left mouse button.

## Version History

In previous releases, the **Ctrl+Plus** and **Ctrl+Minus** keyboard shortcuts zoomed the entire App Designer desktop.

## App Designer: Control color and tab settings in Code View using MATLAB preferences

Color and tab preferences applied to the MATLAB Editor are now also applied to the App Designer Editor.

Additionally, you can now change the background color of the App Designer read-only code. Access this setting in the App Designer tab of MATLAB preferences.

For more information, see [Personalize Code View Appearance](#).

## App Designer: Customize split-screen layouts in the App Designer editor

To view your document in horizontal or vertical split-screen mode in the App Designer **Code View** editor, select a layout in the App Designer toolstrip.

## App Testing Framework: Perform gestures on panels and tables

The app testing framework supports gestures on more UI components:

- Perform `press`, `hover`, and `chooseContextMenu` gestures on panels.
- Perform `choose`, `type`, and `chooseContextMenu` gestures on table UI components.

## App Testing Framework: Close alert dialog box in front of figure window

You can now use the `dismissAlertDialog` method as part of a test case to programmatically close an alert dialog box in front of a figure window. For example, create a modal alert dialog box and close it by calling the method.

```
fig = uifigure;
uiaalert(fig, 'File not found', 'Invalid File')

tc = matlab.uitest.TestCase.forInteractiveUse;
tc.dismissAlertDialog(fig)
```

## Web Apps and Standalone Applications: Datatips supported in graphics

Graphics created in web apps and standalone applications support datatips. Use datatips in these applications just as you would in MATLAB figures.

## Functionality Being Removed or Changed

### GUIDE will be removed in a future release

#### *Warns*

The GUIDE environment and the `guide` function will be removed in a future release.

After GUIDE is removed, existing GUIDE apps will continue to run in MATLAB but will not be editable using the drag-and-drop GUIDE environment. To continue editing an existing GUIDE app and help maintain its compatibility with future MATLAB releases, use one of the suggested migration strategies listed in the table.

App Development	Migration Strategy	How to Migrate
Frequent or ongoing development	Migrate your app to App Designer.	Use the GUIDE to App Designer Migration Tool for MATLAB on <a href="https://www.mathworks.com">mathworks.com</a> .
Minimal or occasional editing	Export your app to a single MATLAB file to manage your app layout and code using MATLAB functions.	Open the app in GUIDE and select <b>File &gt; Export to MATLAB-file</b> .

To create new apps, use App Designer and the `appdesigner` function instead. App Designer is the recommended app development environment in MATLAB.

To learn more about migrating apps, see [GUIDE Migration Strategies](#).

For more information about App Designer, go to [Comparing GUIDE and App Designer on mathworks.com](#).

### GUIDE templates have been removed

All GUIDE templates other than the blank GUI have been removed. To create new apps interactively, use App Designer and the `appdesigner` function instead.

### **Graphics objects and UI components sized using 'points', 'inches', and 'centimeters' units will increase in size on macOS platforms**

*Behavior change in future release*

In a future release, graphics and UI objects that have `Units` or `FontUnits` properties set to `'points'`, `'inches'`, or `'centimeters'` will use a conversion value of 1 pixel = 1/96th inch on macOS platforms. The current conversion value is 1 pixel = 1/72nd inch. As a result, these objects and text elements will display 1.33 times larger than their previous size. This change will provide a more readable default font size and will ensure a consistent object size across Windows and macOS platforms.

The following objects use a unit value of `'pixels'` by default and will not be affected by this change:

- UI components in App Designer or in apps created with the `uifigure` function
- UI components in apps created in GUIDE and migrated to App Designer using the GUIDE to App Designer Migration Tool for MATLAB
- Axes objects created using the `uiaxes` function

For the most control when sizing and laying out your graphics objects and UI components, use a value of `'pixels'` for `Units` and `FontUnits` properties. To maintain object sizes in current and future releases, make these updates in your code:

- Objects with `Units` or `FontUnits` set to `'points'` — Update the value of the property from `'points'` to `'pixels'`.
- Objects with `Units` or `FontUnits` set to `'inches'` — Update the value of the property from `'inches'` to `'pixels'` and multiply all `Position` values by 72.
- Objects with `Units` or `FontUnits` set to `'centimeters'` — Update the value of the property from `'centimeters'` to `'pixels'` and multiply all `Position` values by 72/2.54.

For example, this code creates a push button in a figure window, with its position specified in inches:

```
uicontrol('Units','inches','Position',[0.6 0.1 1.75 0.5]);
```

In a future release, the push button created by this code will display 1.33 times larger on macOS platforms. To maintain the size and position of the push button in current and future releases, update the code to:

```
uicontrol('Units','pixels','Position',[50 10 126 36]);
```

## Performance

### Sparse Matrix Multiplication: Improved performance multiplying large sparse matrices

Matrix multiplication performance has improved when multiplying sparse matrices. The performance improvement arises from added support for multithreading in the operation, and therefore the speedup gets better as the matrix size and number of nonzeros increase.

For example, if you multiply two 1e5-by-1e5 random sparse matrices with approximately two million nonzeros, performance in R2021a is about 4.4x faster than in R2020b on a machine with 6 physical cores.

```
function timingSparseMult
rng default
A = sprand(1e5,1e5,0.0002);
B = sprand(1e5,1e5,0.0002);
tic
C = A*B;
toc
end
```

The approximate execution times are:

**R2020b:** 2.2 s

**R2021a:** 0.5 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSparseMult`.

### Sparse Linear Systems: Improved performance solving sparse linear systems $A*X = B$ with multicolumn B

Solving a linear system of the form  $A*X = B$  by executing  $X = A \setminus B$  shows improved performance when  $A$  is a sparse square matrix and  $B$  is a matrix with two or more columns. The speedup applies to the solving step of the calculation but not the factorization step. The performance improvement arises from added support for multithreading, and therefore the speedup gets better as the number of columns in  $B$  increases.

For example, if you solve  $A*X = B$  using a 1e4-by-1e4 sparse coefficient matrix with approximately 40,000 nonzeros and a  $B$  matrix with 100 columns, performance in R2021a is about 5x faster than in R2020b on a machine with 6 physical cores. This code uses `decomposition` to factor the coefficient matrix, so only the solving process is timed. If you use  $X = A \setminus B$  instead, you still see a speedup, but the time required to factor the matrix is included and has not changed.

```
function timingSparseBackslashMultRHS
rng default
A = sprand(1e4,1e4,0.0003) + speye(1e4);
B = sprand(1e4,100,0.002);
dA = decomposition(A);
tic
x = dA \ B;
```



```
toc
end
```

The approximate execution times are:

**R2020b:** 1.5 s

**R2021a:** 0.3 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSparseBackslashMultRHS`.

## vecnorm Function: Improved performance operating on data with multiple columns

The performance of the `vecnorm` function has improved for all norm types when the data has 16 or more columns and at least  $2^{17}$  elements.

The improvement also applies to N-D array data that can be permuted into a matrix with the requisite number of columns. The speedup varies depending on the type of norm being calculated.

For example, if you calculate the 2-norm of a 1000-by-1000-by-3 array along the third dimension, performance in R2021a is about 7.3x faster than in R2020b.

```
function timingVecnorm
rng default
N = 1000;
A = rand(N,N,3);
for k = 1:200
    D = vecnorm(A,2,3);
end
end
```

The approximate execution times are:

**R2020b:** 8.8 s

**R2021a:** 1.2 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingVecnorm)
```

## ismember Function: Improved performance for cell inputs

The `ismember` function shows improved performance operating on cell inputs. The speedup depends on the size and layout of the data, with the largest speedup when the input has many cells that contain few elements in each cell.

For example, if you use `ismember` to compare two 1000-by-1 cell arrays with 10 elements in each cell, performance in R2021a is about 4.7x faster than in R2020b.

```
function timingIsmember
a = num2cell(char(randi(127,[1000 10])),2);
```

```
b = num2cell(char(randi(127,[1000 10])),2);
tic
for ii = 1:1e4
    ismember(a,b);
end
toc
end
```

The approximate execution times are:

**R2020b:** 6.6 s

**R2021a:** 1.4 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingIsmember`.

## unique Function: Improved performance for numeric, logical, char, and cell inputs

The `unique` function shows improved performance operating on numeric, logical, char, and cell inputs. The speedup generally gets better as the size of the inputs increases, and the improvement applies when using any optional flags except the `'legacy'` flag.

For example, if you operate on a 100-by-1 cell array with 10 elements in each cell, performance in R2021a is about 3.3x faster than in R2020b.

```
function timingUniqueCell
a = num2cell(char(randi(127,[100 10])),2);
tic
for ii = 1:1e5
    b = unique(a);
end
toc
end
```

The approximate execution times are:

**R2020b:** 3.9 s

**R2021a:** 1.2 s

Also, if you use `unique` on a numeric input with 10,000 elements and specify three outputs with the `'stable'` option, performance in R2021a is about 3.5x faster than in R2020b.

```
function timingUniqueNumeric
a = rand(10000,1);
tic
for ii=1:1e4
    [C,ia,ic] = unique(a,'stable');
end
toc
end
```

The approximate execution times are:

**R2020b:** 9.4 s

**R2021a:** 2.7 s

In both cases, the code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the functions `timingUniqueCell` and `timingUniqueNumeric`.

## Graph Functions: Improved performance modifying node and edge lists

`graph` and `digraph` functions that modify the node and edge lists of the graph show improved performance. This applies to the functions `addedge`, `rmedge`, `addnode`, `rmnode`, `subgraph`, and `reordernodes`. The improvement applies to graphs that have no node properties (or only node names), and graphs with no edge properties (or only edge weights). The improvement is most noticeable when one of these functions is called many times in a loop, and the largest improvement applies to graphs that have both node names and edge weights.

For example, if you use `addedge` in a loop to add new edges with node names and edge weights to an empty graph, performance in R2021a is about 13x faster than in R2020b.

```
function timingAddedge
names = string('A':'Z') + (1:10);
names = names(1:100);
rng default
g = graph;
for ii = 1:1e3
    g = addedge(g, names(randi(100)), names(randi(100)), randn);
end
end
```

The approximate execution times are:

**R2020b:** 2.6 s

**R2021a:** 0.2 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by using the `timeit` function:

```
timeit(@timingAddedge)
```

## Axes Toolbar: Appears without delay when axes are ready

Prior to R2021a, when hovering the cursor over figure axes, there was delay before the axes toolbar appeared. Now the toolbar will appear as soon as the axes are ready.

## Rearranging UI Components: Improved performance when rearranging UI components in a UI figure

Programmatically rearranging existing UI components in a figure created with the `uifigure` function shows improved performance.

For example, if you sort 50 panels within a grid layout, performance in R2021a is approximately 1.9x faster than in R2020b.

```

function timingSortComp
% Create components
panels = {};
fig = uifigure;
g = uigridlayout(fig,[1,1], 'RowHeight',40);
g.Scrollable = true;
num = 50;
for i = 1:num
    p = uipanel(g);
    uilabel(p,'Text', ['Panel ', num2str(i)], 'Position', [10 10 70 22]);
    g.RowHeight{end} = 40;
    panels{end+1} = p;
end
drawnow;

% Rearrange components
tic
order = length(panels):-1:1;
for i = 1:length(order)
    panels{i}.Layout.Row = order(i);
end
drawnow;
toc
end

```

The approximate execution times are:

**R2020b:** 0.70 s

**R2021a:** 0.36 s

The code was timed on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system by calling the function `timingSortComp`.

## UI Figure Interactions: Faster responses to scroll, pointer movement, and resize interactions in UI figures

In figures created with the `uifigure` function, the following interactions have improved performance:

- Scrolling in a figure with a `WindowScrollWheelFcn` callback or an object with a predefined scroll behavior
- Resizing a visible figure with a `SizeChangedFcn` callback or an object with a predefined resize behavior
- Moving the mouse pointer in a figure with a `WindowButtonMotionFcn` callback when the figure contains any UI components except axes components

This performance increase is more noticeable when using a trackpad to interact with the figure.

For example, scrolling to zoom in on the plot created by the code below is smoother and more responsive in R2021a than in R2020b.

```

t1 = datetime(2019,1,1);
t2 = datetime(2020,1,1);
dates = linspace(t1,t2,10000);
data = rand(10000,10);
fig = uifigure;
stackedplot(fig,dates,data);

```

On a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system, the responses to the scroll action are:

**R2020b:** When you zoom in on the plot by scrolling for approximately two seconds, the plot has about a five second delay in completing the zoom animation.

**R2021a:** The zoom animation completes immediately after you finish the scroll action.

## Plots in Apps: Improved performance for polar plots, volume visualizations, plots with more than 16 axes, and older systems

Displaying polar plots, volume visualizations, or more than 16 axes in an app have improved performance. The improvement affects plots displayed in apps:

- Plots that are displayed in an app created with App Designer
- Plots displayed in a figure created with the `ui figure` function

Systems with older graphics drivers might experience the improvement for *all* types of plots that are created within the apps and figures listed above. For example, Intel drivers earlier than version 10.0.0.0 for Windows systems will experience additional improvements.

This code creates a polar plot and executes a `for` loop that changes the theta values at every iteration. The `for` loop executes about 2x faster than in R2020b.

```
function timingPolar
f = uifigure;
pax = polaraxes('parent',f);
theta = 0:0.01:2*pi;
rho = sin(2*theta).*cos(2*theta);
pp = polarplot(pax, theta,rho);
pax.FontSize = 12;
drawnow

tic;
for i=1:100
    pp.ThetaData = pp.ThetaData + .02*pi;
    drawnow
end
toc
end
```

The approximate execution times are:

**R2020b:** 10.15 s

**R2021a:** 5.30 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingPolar`.

## Plots in Apps: Improved performance for plots with large numbers of markers

Performance is improved for modifying certain types of plots in apps. You can observe the improvement when the following conditions are true:

- The plots are displayed in an App Designer app, or they are displayed in a figure created with the `uifigure` function.
- Your system is running a locally installed version of MATLAB on a modern Windows or macOS system.
- You run the code either from the MATLAB command window or within a program file (.m file).

Note that plots created in live scripts do not show this improvement.

The plots typically contain large numbers of markers, and your code updates an aspect of those markers, such as their positions. The improvement is more significant as you increase the number of markers. For example, if you create a scatter plot with 10 million markers, and change the marker positions 10 times, the performance in R2021a is about 1.3x faster than in R2020b.

```
function timingScatter
f = uifigure;
a = axes(f);
x = rand(1e7, 2);
s = scatter(a,x(:,1),x(:,2),'Marker','*');
drawnow;

tic;
for i=1:10
    x = rand(1e7,2);
    s.XData = x(:,1);
    s.YData = x(:,2);
    drawnow
end
toc
end
```

The approximate execution times are:

**R2020b:** 19.43 s

**R2021a:** 14.88 s

The code was timed on a macOS 10.14.6, Intel Core i9 CPU @ 3.60 GHz test system by calling the function `timingScatter`.

## Live Editor: Improved performance when saving large live scripts or functions

When saving large live scripts or functions, you can continue using the Live Editor sooner in R2021a than in R2020b. While you continue to use the Live Editor, MATLAB saves the file in the background. When MATLAB finishes saving the file, the asterisk (\*) next to the file name disappears, indicating that the file is saved.

For example, on a Windows 10, Intel Xeon E5-1650 CPU @ 3.60 GHz test system, if you save a live script containing 35,000 lines of code and then click the title bar of another document open in the

Live Editor, MATLAB immediately switches to the other open document. In R2020b, there is a noticeable delay before MATLAB switches to the other open document.

## Software Development Tools

### Projects: List all referenced projects of the current project

You can now use `listAllProjectReferences` to programmatically list all projects in the reference hierarchy of a specified project.

### Projects: List impacted project files

You can now use `listImpactedFiles` to programmatically list all project files impacted by changes to a specified file.

### Dependency Analyzer: Find required add-ons

Starting in R2021a, the Dependency Analyzer detects and lists required add-ons, including apps and toolboxes, for the whole project or for selected files. The Dependency Analyzer might not detect required support packages. For more details, see [Find Required Products and Add-Ons](#).

### Unit Testing Framework: Create test runners using alternative syntax

You can now use the `testrunner` function to create a runner for tests authored using the MATLAB unit testing framework or Simulink® Test™. In previous releases, you can explicitly create a runner only by calling one of the static methods of the `matlab.unittest.TestRunner` class.

Use the `testrunner` function to create a default runner, a minimal runner with no plugins installed, or a runner configured for text output. For example, create a default runner to run the tests in a test class.

```
suite = testsuite('MyTestClass');  
runner = testrunner;  
results = run(runner,suite);
```

### Unit Testing Framework: Initialize parameterization properties at suite creation time

Starting in R2021a, you can specify parameterization properties that do not have a default value. This feature is useful when parameters cannot be determined at the time MATLAB loads the test class definition. To initialize a parameterization property at test suite creation time, use a static method with the `TestParameterDefinition` attribute. For more information, see [Define Parameters at Suite Creation Time](#).

### Unit Testing Framework: Run tests in parallel on thread-based pool

You can now run your tests on a thread-based parallel pool (requires Parallel Computing Toolbox). To run tests using thread workers, start a thread-based pool and then call the `runInParallel` method or the `runtests` function with the `UseParallel` name-value pair argument.

Thread-based parallel pools support only a subset of MATLAB and testing framework functionality. For more information, see `runInParallel` or `runtests`.



## Unit Testing Framework: Run tests in MATLAB Online interactively

Starting in R2021a, you can run tests in MATLAB Online interactively. When you open a .m file defining a function-based or class-based test in MATLAB Online, the toolstrip lets you run all tests in the file or run the current test in the file. Also, you can customize the test run with options, such as running tests in parallel (requires Parallel Computing Toolbox) or running tests with a specified level of output detail.

The **Run Tests** section in the **Editor** tab of the toolstrip provides an alternative to programmatically running tests with the `runtests` function. For more information, see [Run Tests in Editor](#).

## App Testing Framework: Perform gestures on panels and tables

The app testing framework supports gestures on more UI components:

- Perform `press`, `hover`, and `chooseContextMenu` gestures on panels.
- Perform `choose`, `type`, and `chooseContextMenu` gestures on table UI components.

## App Testing Framework: Close alert dialog box in front of figure window

You can now use the `dismissAlertDialog` method as part of a test case to programmatically close an alert dialog box in front of a figure window. For example, create a modal alert dialog box and close it by calling the method.

```
fig = uifigure;
UIAlert(fig, 'File not found', 'Invalid File')

tc = matlab.uitest.TestCase.forInteractiveUse;
tc.dismissAlertDialog(fig)
```

## Functionality being removed or changed

### Character data is not supported in custom examples demos.xml file

*Behavior change*

Starting in R2021a, when creating custom examples, character data is not supported in the description of the `demos.xml` file. If you have a `demos.xml` file that contains character data such as `&lt;`, `&gt;`, `&apos;`, `&quot;`, and `&amp;`, the description does not appear correctly in the Help browser.

To patch an existing `demos.xml` that contains character data, use the `patchdemoxmlfile` function. `patchdemoxmlfile` patches the specified `demos.xml` file, replacing character data with non-character data.

For example, patch the `demos.xml` file in the folder `D:\Work\mytoolbox\help`:

```
patchdemoxmlfile D:\Work\mytoolbox\help
```

## External Language Interfaces

### C++ Interface: Support for C++ language features

The C++ interface supports these additional C++ language features.

- Support for `std::vector` values containing `std::string` values and C++ arrays containing C-style strings. For more information, see [String and Character Types in the C++ to MATLAB Data Type Mapping](#) topic.
- Support for `void*` values as input and output arguments. For more information, see [Use void\\* Arguments](#), [void\\* Argument Types](#), and [addOpaqueType](#).
- Pass C++ functions to function pointers. For more information, see [Use Function Type Arguments](#) and [addFunctionType](#).
- Support for function and member function template instantiations. Publishers can modify function names. For more information, see [Customize Function Template Names](#).

### C++ Interface: Publisher options and analysis

The C++ interface supports these build configuration features.

- Generate an interface from header and source (`.cpp`) files. Pass a `.cpp` or `.hpp` file in the `clibgen.generateLibraryDefinition SourceFiles` argument.
- Generate an interface from a header and a `.dll` file for Microsoft Visual Studio compilers. Pass a `.dll` file in the `clibgen.generateLibraryDefinition LibraryFiles` argument.
- Improved troubleshooting messages.

### Java Packages to be removed

Java packages and subpackages that currently ship with MATLAB will not be available in MATLAB in a future release.

### Version History

To continue using a Java package, install its JAR file and add the JAR file to the static path in MATLAB using the instructions in [Static Path](#).

### Java Engine: MATLAB value object support

To work with MATLAB value objects in a Java engine application, use the `com.mathworks.matlab.types.ValueObject` class in the [Java Engine API Summary](#). You can create a value object in MATLAB, return it to Java, and call its methods. For information about mapping Java data types to MATLAB data types, see [Java Data Type Conversions](#).

### Python Interface and Engine: Version 3.6 support discontinued

Support for Python version 3.6 is discontinued.

## Version History

To ensure continued support for your applications, upgrade to a supported version of Python, either version 3.7 or 3.8. For more information, see [Versions of Python Compatible with MATLAB Products by Release](#).

## Perl 5.32.0: MATLAB support on Windows

As of R2021a, MATLAB on Windows ships with an updated version of Perl, version 5.32.0.

- See [www.perl.org](http://www.perl.org) for a standard distribution of Perl, Perl source, and information about using Perl.
- See <https://metacpan.org/pod/HTML::Parser> for a standard distribution of `HTML::Parser`, source code, and information about using `HTML::Parser`.
- See <https://metacpan.org/pod/HTML::Tagset> for a standard distribution of `HTML::Tagset`, source code, and information about using `HTML::Tagset`.

## Version History

If you use the `perl` command on Windows platforms, see [www.perl.org](http://www.perl.org) for information about using this version of the Perl programming language.

## Hardware Support

### Support added for IMU sensors

The MATLAB Support Package for Raspberry Pi Hardware now provides code generation and connected IO support to Raspberry Pi functions for the these IMU sensors:

- HTS221
- LPS22HB
- LSM303C
- LSM6DSL
- LSM9DS1
- MPU-6050
- MPU-9250

### New functionalities added to Raspberry Pi Resource Monitor app

The Raspberry Pi Resource Monitor App from the MATLAB Support Package for Raspberry Pi Hardware has been improved to:

- Display peripherals used in a MATLAB or Simulink application deployed on the Raspberry Pi hardware
- Enable or disable peripherals
- Check for missing libraries and packages
- Display all processes currently running on the Raspberry Pi hardware

# R2020b

---

**Version: 9.9**

**New Features**

**Bug Fixes**

**Version History**

## Environment

### **MATLAB Online Accessibility: Use a screen reader to interact with the Command Window and create scripts and functions**

In MATLAB Online, you can use a screen reader to interact with the Command Window, create and edit scripts and functions in the Editor, and navigate through the MATLAB desktop tools. Using a screen reader is not supported in the Live Editor.

For more information, see [Use a Screen Reader in MATLAB Online](#).


### **Live Editor Images: Add alternative text to images**

You can add alternative text to an image in a live script or function to make it accessible to individuals using a screen reader. To add alternative text, right-click the image and select **Edit Image...** from the context menu. Add text to the **Alt Text** edit field.

### **Live Editor Images: Change the size of images**

To change the size of an image in a live script or function, right-click the image and select **Edit Image...** from the context menu. Then, to specify a size relative to the original image size, select **Relative (%)** and specify the width and height of the image as a percentage of the original image. To specify an absolute size, select **Absolute (px)** and specify the width and height of the image in pixels. To return to the original image size, right-click the image and select **Reset Image**.

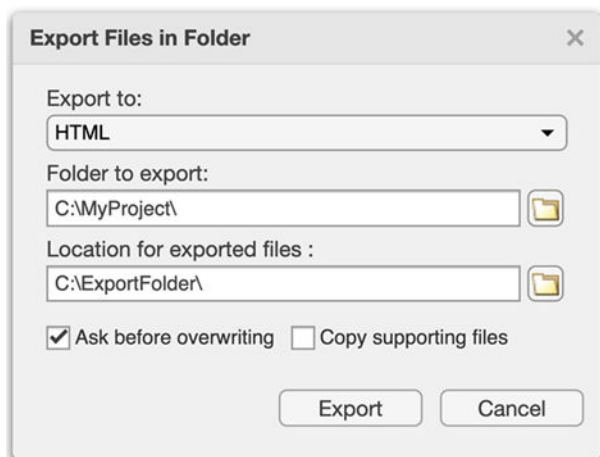
### **Live Editor Hyperlinks: Navigate to existing files from a live script or live function using links**

Use hyperlinks to navigate to existing files from a live script or live function. To insert a hyperlink to an existing file, select the text to link in the current file, go to the **Insert** tab, and click  **Hyperlink**. Edit your display text (optional), select **Existing File**, and then enter or browse for the file path.

For more information about adding hyperlinks into live scripts and live functions, see [Format Files in the Live Editor](#).

### **Live Editor Export: Export all live scripts and live functions in a folder to a standard format**

You can export all of the live scripts and live functions in a folder to a standard format. Available formats include PDF, Microsoft Word, HTML, and LaTeX. To export the contents of a folder, on the **Live Editor** tab, select **Save > Export Folder...** Then, specify the export format, the location of the folder to be exported, and the location for the exported files.



For more information about exporting live scripts and live functions, see [Share Live Scripts and Functions](#).

## matlabRelease Object: Query MATLAB Release Information

`matlabRelease` is a new object that makes it possible to quickly query the MATLAB information of your system. `isMATLABReleaseOlderThan` will allow you to determine if your current MATLAB release is older than a specified release. Use these functions to quickly verify release versions.

## Query Parallel Functionality: Determine if support for Parallel Computing Toolbox functionality is available

You can now query if support for GPU and parallel pool functionality is available in your MATLAB installation using the following functions:

- `canUseGPU`
- `canUseParallelPool`

Use these functions to check supported functionality and avoid executing code that relies on specific hardware constraints.

## Comparison Tool: Compare text files in MATLAB Online

Starting in R2020b, you can compare text files in MATLAB Online using the `visdiff` function. To compare text files, in the Command Window, type:

```
visdiff('filename1.m', 'filename2.m');
```

## Language and Programming

### pattern Object and Functions: Match patterns in text functions

Patterns are an intuitive alternative to regular expressions for matching patterns in text. Pattern functions can be combined together in expressions in order to build complex patterns that can then be used as inputs for text-searching functions. For instance, to define a pattern for MATLAB release names, which start with "R", followed by the four-digit year, and then either "a" or "b":

```
pat = "R" + digitsPattern(4) + ("a"|"b");
```

Match that pattern in a string:

```
str = "String arrays were introduced in R2016b. Patterns were added in R2020b.";
extract(str,pat)
```

```
ans =
    2x1 string array
    "R2016b"
    "R2020b"
```

For more information, see `pattern`.

### extract Function: Extract substrings from strings

To extract substrings from strings, use the `extract` function. You can specify the substring to be extracted as literal text or by using a pattern object to match the text in a substring.

### Functions: New validation functions for arguments and properties

The following functions are designed for use in function argument and property validation.

- `mustBeA` Validate that value comes from one of specified classes
- `mustBeNonmissing` Validate that value is not missing
- `mustBeFloat` Validate that value is floating-point array
- `mustBeScalarOrEmpty` Validate that value is scalar or empty
- `mustBeVector` Validate that value is vector
- `mustBeInRange` Validate that value is in the specified range
- `mustBeFile` Validate that path refers to file
- `mustBeFolder` Validate that input path refers to folder
- `mustBeValidVariableName` Validate that input name is valid variable name
- `mustBeText` Validate that value is a string array, character vector, or cell array of character vectors
- `mustBeTextScalar` Validate that value is a single piece of text
- `mustBeNonzeroLengthText` Validate that value is string array, character vector, or cell array of character vectors that has non-zero length



## underlyingType, isUnderlyingType, and mustBeUnderlyingType Functions: Query the underlying data type of classes

The `class` function is useful to determine the class of a variable. However, some classes in MATLAB can contain underlying data that has a different type compared to what `class` returns. Example classes include `gpuArray`, `dlarray`, and `distributed` arrays. The `underlyingType`, `isUnderlyingType`, and `mustBeUnderlyingType` functions now provide a simple way to query the underlying data types of those classes.

For most classes, `class(X)` and `underlyingType(X)` return the same answer. However, for classes that can contain underlying data of a different type, the return values are different. For a `gpuArray X` that contains data of type `double`, for example, `class(X)` returns `'gpuArray'`, whereas `underlyingType(X)` returns the underlying MATLAB data type, `'double'`.

## height and width Functions: Return number of rows or columns in an array

The `height` and `width` functions now work with arrays in addition to tables. `height` returns the number of rows in the array and `width` returns the number of columns.

## Class conversions: Assignment operations convert more classes into built-in data types

In assignment statements such as `A(1:k) = C`, where `A` has a built-in data type such as `double`, MATLAB attempts to convert `C` to be the same data type as `A` using a series of conversions. That conversion behavior has changed.

- Previously, only one conversion was attempted. This conversion attempted to use a class conversion function, such as `logical(C)` or `double(C)`, to obtain a variable with the same data type as `A`.
- Now, if the first conversion fails, or if it returns a type that differs from `A`, then MATLAB attempts a secondary conversion with `cast(C, "like", A)`. If `C` belongs to a class that defines a `cast` method that supports the "like" flag, and that method returns a value with the same type as `A`, then MATLAB uses the resulting value to perform the assignment into `A`.

If both conversion attempts fail, then MATLAB throws a `MATLAB:invalidConversion` error.

An example of code that used to error, but now executes, is

```
a = 1:6;
g = distributed([11 12 13]);
a(1:3) = g

a =

    11    12    13     4     5     6
```

## Version History

Some assignment statements that used to throw an error now execute. If your code relied on the errors that MATLAB threw for those conversions, such as within a `try/catch` block, then your code might no longer catch those errors.

## Functionality being removed or changed

### **hex2dec, bin2dec, and base2dec Functions: Issue warning when inputs are greater than or equal to flintmax**

#### *Behavior change*

Starting in R2020b, the `hex2dec`, `bin2dec`, and `base2dec` functions issue a warning when their inputs have values greater than or equal to `flintmax`. In previous releases, these functions did not issue a warning when their inputs had such values.

These functions now issue a warning because inputs representing integers greater than or equal to `flintmax` might not be represented exactly as double-precision floating-point values.

To convert values greater than `flintmax` exactly, consider one of these alternatives:

- In place of scalar text inputs, use hexadecimal or binary literals representing the same values. When you write a value as a literal, MATLAB stores it as an integer that represents the value exactly. For more information, see [Hexadecimal and Binary Values](#).
- To convert hexadecimal inputs greater than `flintmax`, you can use the `sscanf` function with the `%lx` operator. When you use `%lx`, the converted values are integers that have the `uint64` data type. These integers have enough storage to represent values greater than `flintmax` exactly.

### **Defining classes and packages: Using `schema.m` will not be supported in a future release**

#### *Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

## Data Analysis

### Implicit Expansion: For calendarDuration, categorical, datetime, and duration arrays, automatically expand dimensions of length 1 when applying element-wise operations and functions

calendarDuration, categorical, datetime, and duration arrays now support implicit expansion for certain operations and functions. In previous releases, these operations and functions only supported implicit expansion for numeric and logical arrays.

Implicit expansion is a generalization of scalar expansion. With scalar expansion, a scalar expands to be the same size as another array to facilitate element-wise operations. With implicit expansion, the element-wise operators and functions listed in the table can implicitly expand their inputs to be the same size, as long as the arrays have compatible sizes. Two arrays have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of them is 1. See Compatible Array Sizes for Basic Operations and Array vs. Matrix Operations for more information.

For each operator or function, the table shows the data types that gained support for implicit expansion in R2020b.

Operator or Function	Data Types with New Support for Implicit Expansion in R2020b
+ -	calendarDuration, datetime, and duration
.*	calendarDuration, categorical, and duration
./ .\	duration
.^	None
== ~=	categorical, datetime, and duration
< <= > >=	categorical (ordinal arrays only), datetime, and duration
max min	categorical (ordinal arrays only), datetime, and duration
mod rem	duration
isbetween	datetime and duration

## Version History

If your code uses any of the element-wise operators or functions listed above and relies on the errors that MATLAB returns from mismatched sizes, particularly within a `try/catch` block, then your code might no longer catch those errors. This change is because some combinations of input sizes that previously caused errors are now valid.

For example, in older releases of MATLAB, you could not add a row and a column vector that are `duration` vectors, but those operands are now valid for addition with implicit expansion. In other words, an expression like `seconds([1 2]) + seconds([1; 2])` previously returned a size mismatch error, but now it executes.

### **normalize Function: Scale data by interquartile range**

You can use the `'medianiqr'` method with the `normalize` function to center data on a median of 0 and scale it to have an interquartile range of 1. Additionally, when using the `'scale'` method, the new `'iqr'` option allows you to scale the data by the interquartile range.

### **groupsummary Function: Summarize data using functions that require multiple input arguments**

When specifying a summary computation, `groupsummary` now supports function handles and anonymous functions that require more than one input argument.

### **fillmissing Function and Clean Missing Data Live Editor Task: Specify maximum gap size to fill**

When filling missing data using the `fillmissing` function, you can use the `'MaxGap'` name-value pair to fill only gaps of missing data up to a specified size. When using the **Clean Missing Data** Live Editor task, enter a gap size next to **Max gap to fill**.

### **Clean Outlier Data Live Editor Task: Define outliers based on percentile thresholds**

The **Clean Outlier Data** Live Editor task now offers the option to define outliers in data as points outside of a percentile range. To use this option, select **Percentiles** as the **Detection method**, and specify values for the **Lower threshold** and **Upper threshold**.

## Functionality being removed or changed

### **table and timetable Data Types: Change to dimension name selection when combining or joining tables and timetables**

*Behavior change*

In R2020b, when you combine tables and timetables, the dimension names of the output table come from the first nondefault dimension names in the input table and timetables. In previous releases, the dimension names always came from the first input, even when that table or timetable had default dimension names.

For example, if TT1 and TT2 are timetables, and TT2 has the nondefault name 'Start' as the name of its first dimension, then when you concatenate the timetables the output also has 'Start' as the name of its first dimension. The name of the first dimension of TT1 is 'Time', but that is a default dimension name. In previous releases, the name of the first dimension of the output was 'Time' because that name was in the first input timetable.

```
TT1 = timetable(seconds(1:2)',[1;2]);
TT2 = timetable(seconds(1:2)',[3;4]);
TT2.Properties.DimensionNames{1} = 'Start';
TT = [TT1 ; TT2]
```

TT =

4×1 timetable

Start	Var1
1 sec	1
2 sec	2
1 sec	3
2 sec	4

This change in behavior affects the functions in this list.

- `cat`
- `horzcat`
- `innerjoin`
- `outerjoin`
- `setxor`
- `union`
- `vertcat`

### **Retime Timetable Live Editor Task: Use linear interpolation as default general rule for adjusting data**

*Behavior change*

In R2020b, linear interpolation is the default general rule for adjusting data in the **Retime Timetable** live editor task. In previous releases, the default general rule is to fill gaps in the output timetable with missing values.

However, even in R2020b the default rule reverts to filling gaps with missing values if either of these conditions is met:

- The row times of the input timetable are not sorted.
- The input timetable has at least one variable whose data type is not numeric, `datetime`, or `duration`.

## Data Import and Export

### **readstruct and writestruct functions: Read and write structured data in XML files**

Read and write structured data stored in XML files using the `readstruct` and `writestruct` functions:

- Use the `readstruct` function to read structured data stored as XML files.
- Use the `writestruct` function to write MATLAB structures to XML files.

### **readlines function: Read the lines in a text file as a string array**

Use the `readlines` function to read each line in a text file as an N-by-1 string array, where N is the number of lines in the text file. For more information, see `readlines`.

### **Spreadsheet files: Customize formatting when writing data to spreadsheet files with PreserveFormat and AutoFitWidth**

The `writetable`, `writematrix`, and `writetimetable` functions have two new name-value pairs that enable you to format spreadsheet files: `'PreserveFormat'` and `'AutoFitWidth'`.

- `'PreserveFormat'` specifies whether to preserve the existing cell formatting of the original data.
- `'AutoFitWidth'` specifies whether to automatically adjust the column width of the spreadsheet file to fit the data.

### **imread function and Tiff object: Read images from Aperio SVS and TIFF files containing JPEG2000 compression**

The `imread` function and the `Tiff` object can now read Aperio SVS microscopy image files and TIFF image files with JPEG2000 compression.

### **ArrayDatastore object: Create datastores from in-memory data**

The `arrayDatastore` function creates a datastore from in-memory data. You can combine `ArrayDatastore` objects with datastores that contain on-disk data (such as `ImageDatastore` and `TabularTextDatastore` objects) using the `combine` and `transform` functions. To choose the ways that an `ArrayDatastore` object reads and returns data, specify these properties: `'IterationDimension'`, `'ReadSize'`, and `'OutputType'`. For more information, see `arrayDatastore`.

### **Datastore: Transform multiple datastores using the transform function**

The `transform` function now accepts multiple datastores as input. Specify each datastore as another input argument.

```
ds = transform(@fcn, ds1, ds2, ds3);
```

The `transform` function creates one `TransformedDatastore` object from the resulting transformation. For more information on the `TransformedDatastore` object, see `TransformedDatastore`.

## **FileDatastore object: Shuffle and create subsets of a FileDatastore**

You can use the `shuffle` and `subset` functions to shuffle and create subsets of a `FileDatastore` object. You must use the `'ReadMode', 'file'` name-value pair argument to use `subset` and `shuffle` on a `FileDatastore` object.

## **writeall function: Write data from text and spreadsheet files to different row groups in Parquet files**

In R2020b, when writing to parquet files using the `writeall` function, you do not need to set the `ReadSize` property of `TabularTextDatastore`, `SpreadsheetDatastore`, or `ParquetDatastore` objects to `'file'`. `writeall` will write the amount of data specified by the `ReadSize` property of the datastore to a separate row group in the Parquet file.

## **fileparts function: Parse file names specified as cell arrays of character vectors and string arrays**

The `fileparts` function now accepts cell arrays of character vectors and string arrays as the input file names.

## **Audio devices: Refresh the available audio devices using the audiodevreset function**

You can use the `audiodevreset` function to refresh the available audio devices after adding or removing a device from your machine. Use the `audiodevinfo` function to view the updated list of audio devices.

## **Audio files and web-based data: Read and write remotely stored audio files using audioread, audiowrite, and audioinfo**

You can access audio files stored in remote locations, such as Amazon S3, Windows Azure Blob Service, and HDFS™, using the `audioread`, `audiowrite`, and `audioinfo` functions.

When reading data from a remote location, you must specify the full path using a uniform resource locator (URL). For example, read a `.wav` file from the Amazon S3 cloud.

```
audio = audioread('s3://bucketname/path_to_file/sample_audio.wav');
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## HDF5 files and web-based data: Read and write remotely-stored HDF5 files using existing HDF5 functions

MATLAB HDF5 functions can now work with remote files. Use the existing high-level and low-level functions to read and write files stored in remote locations, such as Amazon S3, Windows Azure Blob Service, and HDFS.

When reading data from a remote location, you must specify the full path using a uniform resource locator (URL). For example, display the metadata of an HDF5 file from the Amazon S3 cloud.

```
h5disp('s3://bucketname/path_to_file/my_file.h5');
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## HDF5 files: Read and write file names encoded using Unicode characters

You can now read and write file names encoded as Unicode characters using the high-level and low-level HDF5 functions in MATLAB.

## Scientific File Format Libraries: NetCDF library upgraded to 4.7.3

The NetCDF library is upgraded to version 4.7.3.

## Image File Format Libraries: LibTIFF library upgraded to version 4.1.0

The LibTIFF library is upgraded to version 4.1.0.

## Bluetooth Interface: Support for communicating with Bluetooth devices

You can now use MATLAB to connect to Bluetooth devices and transmit and receive ASCII and binary data.

Get started with the new interface by viewing a list of all Bluetooth devices on your computer using `bluetoothlist`. In this example, an HC-06 Bluetooth module is paired to the computer.

```
list = bluetoothlist
```

```
list=1x4 table
      Name          Address      Channel      Status
  _____  _____  _____  _____
    "HC-06"      "98D331FB3B77"      1      "Ready to connect"
```

Then, create a `bluetooth` object, write data to the device, and read from it. In this example, the HC-06 is configured as a loopback device.

```
b = bluetooth("HC-06");
write(b,1:10);
read(b,10);
```

For more information, see [Bluetooth Communication](#).



## TCP/IP Client Interface: New functions and properties

The TCP/IP client interface has a new set of functions and properties.

You can still perform the following operations using existing functions:

- Create a TCP/IP client connection with a TCP/IP server using the `tcpclient` function.
- Read data from a remote host using the `read` function.
- Write data to a remote host using the `write` function.

You can now perform the following operations using new functions:

- Start an echo TCP/IP server using the `echotcpip` function.
- Read a line of ASCII string data from a remote host using the `readline` function.
- Write a line of ASCII string data to a remote host using the `writeline` function.
- Set a terminator for ASCII string communication with a remote host using the `configureTerminator` function.
- Set a callback function and trigger condition for communication with a remote host using the `configureCallback` function.
- Flush buffers for communication with a remote host using the `flush` function.

Get started with the TCP/IP client interface by creating a `tcpclient` object connected to a TCP/IP echo server, writing data to it, and reading from it.

```
echotcpip("on",3030)
t = tcpclient("localhost",3030)
write(t,1:5,"uint8")
read(t,5);
```

For more information, see TCP/IP Communication.

## Serial Port Interface: Improved performance

The `serialport` interface shows improved performance over the `serial` interface. For example, this code for writing and reading data with the `serialport` object is about 1.1x faster than the code for writing and reading data with the `serial` object with the default baud rate of 9600.

```
% s is a serial object
function timingTest(s,bytecount)
fwrite(s,1:bytecount,"uint8");
fread(s,bytecount,"uint8");
end

% s is a serialport object
function timingTest(s,bytecount)
write(s,1:bytecount,"uint8");
read(s,bytecount,"uint8");
end
```

The approximate execution times for different baud rates follow:

	s.BaudRate
--	------------

	9600	19200	56000	115200
serial	120 ms	68 ms	31 ms	23 ms
serialport	109 ms	55 ms	21 ms	11 ms

The code was timed on a Windows 10, Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60 GHz test system using the `timeit` function:

```
bytecount = 100;
timeit(@()timingTest(s,bytecount))
```

The tests were done using a serial loopback connector.

For more information, see [Serial Port Devices](#).

## Functionality being removed or changed

### The 'PreserveVariableNames' property of `readtable` is no longer recommended

*Behavior change*

The 'PreserveVariableNames' property of `readtable` has been removed. Use the 'VariableNamingRule' name-value pair instead. Specify one of the following values:

- 'preserve' preserves variable names that are not valid MATLAB identifiers, such as variable names that include spaces and non-ASCII characters.
- 'modify' converts invalid variable names to valid MATLAB identifiers.

### The `UnderlyingDatastore` property of `TransformedDatastore` is no longer recommended

*Behavior change*

The `UnderlyingDatastore` property of `TransformedDatastore` has been removed. Use the `UnderlyingDatastores` property instead. Calling the `UnderlyingDatastores` property returns a cell array containing zero or more datastores. For more information, see [TransformedDatastore](#).

### The `partition` function adds remaining observations to the first existing partitions

*Behavior change*

If you specify a number of partitions in the `partition` function that is not a numerical factor of the number of files in the datastore, the `partition` function will place each of the remaining observations in the existing partitions, starting with the first partition.

The number of existing partitions that contain an additional observation is equal to the remainder obtained when dividing the number of files in the datastore by the number of partitions. For example, if your datastore object contains 23 files that you wish to partition into 3 parts, the first two partitions that `partition` creates will contain 8 files, and the last partition will contain 7 files.

### Server certificate and hostname validation are enabled by default when accessing OPeNDAP servers with the NetCDF interface

*Behavior change*

In R2020b, the MATLAB NetCDF interface connects only to trusted data access protocol (DAP) endpoints by default. Previously, when you accessed an OPeNDAP server, both the server certificate and hostname validation were disabled by default.

To learn how to disable server certificate and hostname validation when accessing an OPeNDAP server, see [Import NetCDF Files and OPeNDAP Data](#).

### **isSingleReadPerFile has been removed**

The `isSingleReadPerFile` function of the `matlab.io.datastore.FileWritable` class has been removed.

## Mathematics

### **Optimize Live Editor Task: Solve optimization problems interactively**

The **Optimize** Live Editor task lets you optimize multivariable functions or solve scalar equations interactively. The task provides a visual way to access the `fminbnd`, `fminsearch`, `fzero`, and `lsqnonneg` solvers and set their options. For an example, see [Optimize Live Editor Task](#).

### **pagetimes Function: Perform matrix multiplication on pages of N-D arrays**

Use the `pagetimes` function to perform batched matrix multiplication on the pages of N-D arrays. In this context, the N-D array is treated as a container for several 2-D matrices.

### **pagetranspose and pagectranspose Functions: Transpose pages of N-D arrays**

Use the `pagetranspose` and `pagectranspose` functions to transpose the pages of an N-D array. In this context, the N-D array is treated as a container for several 2-D matrices.

### **svdsketch Function: Compute SVD factors of low-rank matrix sketch**

`svdsketch` computes the singular value decomposition (SVD) factors of a low-rank sketch of the input matrix. This operation preserves the most important features of the matrix based on a specified tolerance. For large matrices, `svdsketch` can typically compute an approximate SVD faster than `svds`.

## **Functionality being removed or changed**

### **'cubic' method of `interp1` performs cubic convolution**

#### *Behavior change*

In R2020b, the 'cubic' interpolation method of `interp1` performs cubic convolution. The 'v5cubic' and 'cubic' interpolation methods now perform the same type of interpolation, which is consistent with the behavior of `interp2`, `interp3`, and `interpn`. The cubic convolution interpolation method is intended for uniformly-spaced data, and it falls back to 'spline' interpolation for irregularly-spaced data.

In previous releases, 'cubic' was the same as 'pchip', and only 'v5cubic' performed cubic convolution.

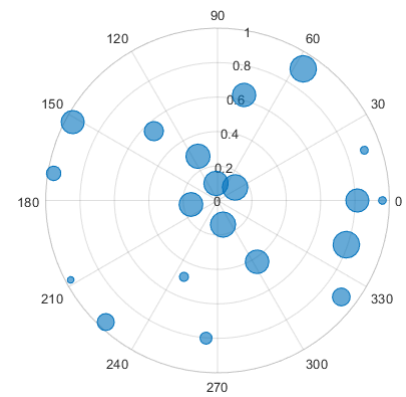
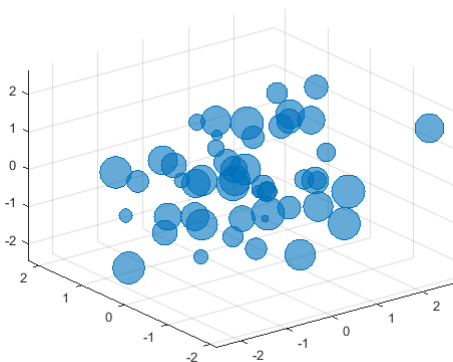
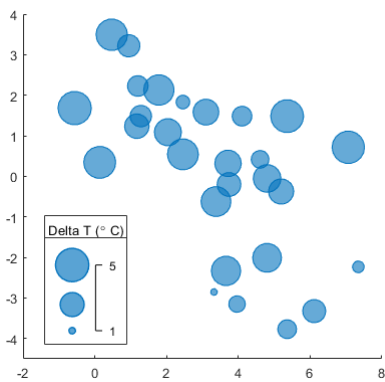
## Graphics

### **bubblechart, bubblechart3, and polarbubblechart Functions: Create bubble charts in 2-D, 3-D, and in polar coordinates**

Use the `bubblechart` and `bubblechart3` functions to create bubble charts in 2-D and 3-D Cartesian spaces, respectively. Use the `polarbubblechart` function to create bubble charts in polar coordinates.

Use the following functions to customize different aspects of your bubble charts:

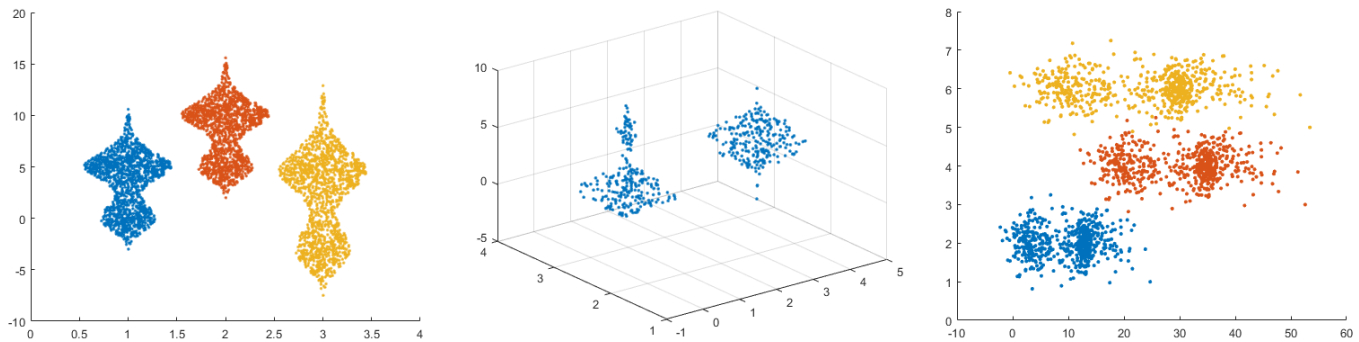
- `bubblelegend` function — Illustrate the correspondence between the bubble sizes and the data values.
- `bubblelim` function — Control the correspondence between the relative bubble sizes and the data values.
- `bubblesize` — Control the absolute bubble sizes in points, where one point equals 1/72 inch.



### **Swarm charts and Scatter objects: Visualize distributions of discrete data**

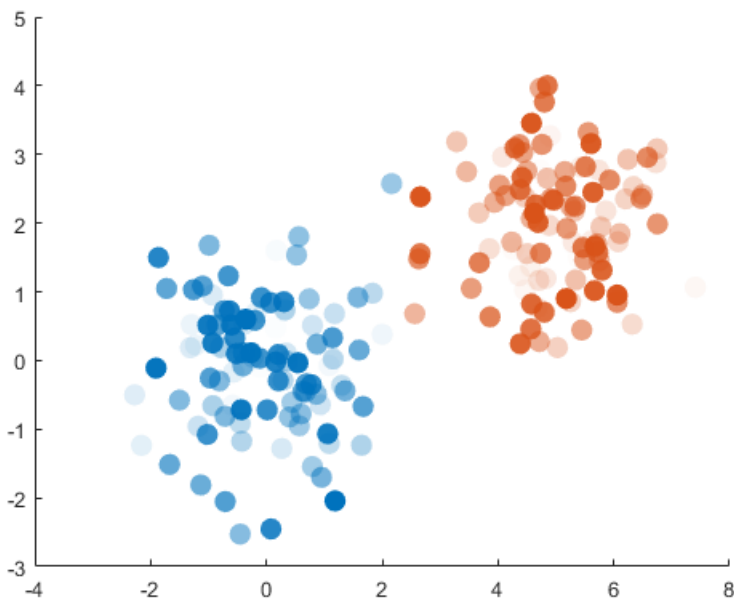
Use the `swarmchart` function to visualize distributions of scattered points at discrete values of X. Use the `swarmchart3` function to visualize distributions for combinations of discrete X and Y. Both functions return Scatter objects that have these new properties:

- `XJitter`, `YJitter`, and `ZJitter` — Control the algorithm for spreading (jittering) the points along the x-, y-, or z-dimensions.
- `XJitterWidth`, `YJitterWidth`, and `ZJitterWidth` — Control the maximum amount of jitter.



## scatter Function: Vary the transparency across all points

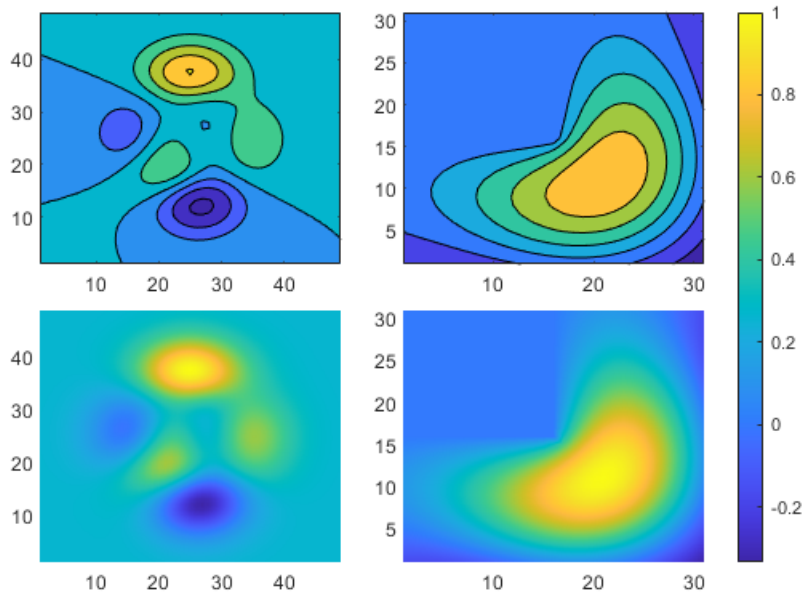
When you create a scatter plot using the `scatter` function, you can vary the level of transparency across the points by setting the `AlphaData` and `AlphaDataMapping` properties of the `Scatter` object.



## tiledlayout and nexttile Functions: Improved placement of legends, and colorbars, and shared decorations

### Improved Legend and Colorbar Placement

- You can now place shared legends, shared colorbars, or additional axes into any of four outer tiles around the perimeter of a tiled chart layout. For more information, see `nexttile`.



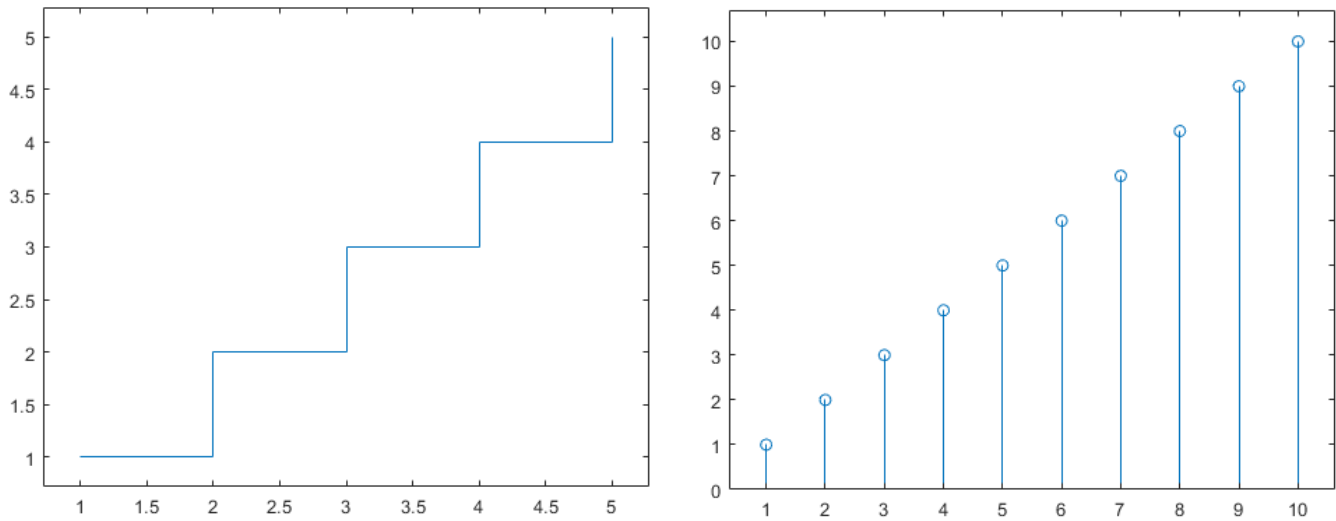
- Tiled chart layouts also provide more support for positioning legends. When you add a legend to a plot, the layout provides better support for the 'best', 'bestoutside', and 'none' location options.

#### More Consistent Layout for Shared Decorations and Polar Axes

- When you include shared decorations (such as shared titles and colorbars) and manually adjust the axes aspect ratios, the presentation is better and more consistent with other layouts.
- The presentation is also better for layouts containing polar axes with the 'flow' tile arrangement.

#### axis Function: Pad axis limits to show plotted data near the limits more clearly

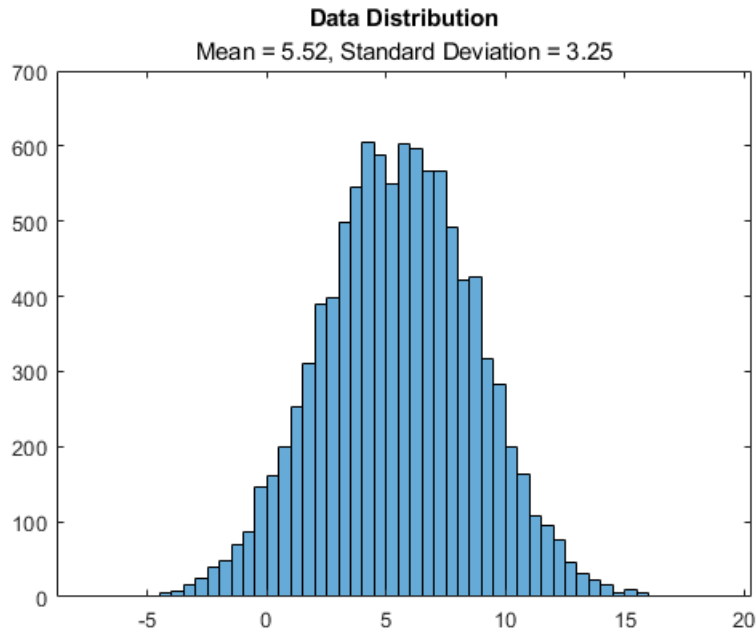
Specify the 'padded' option when calling the axis function to pad the upper and lower limits of the x- and y-axis. This option is helpful when your plot displays data near one or more of the axis limits.



## Titles, Subtitles, and Axis Labels: Add subtitles to plots, and align titles and axis labels with the plot box

### Create Subtitles

- Create a title and subtitle for a plot by calling the `title` function with two character vector or string arguments.
- For more control over the appearance of each text object, call the `title` and `subtitle` functions separately.



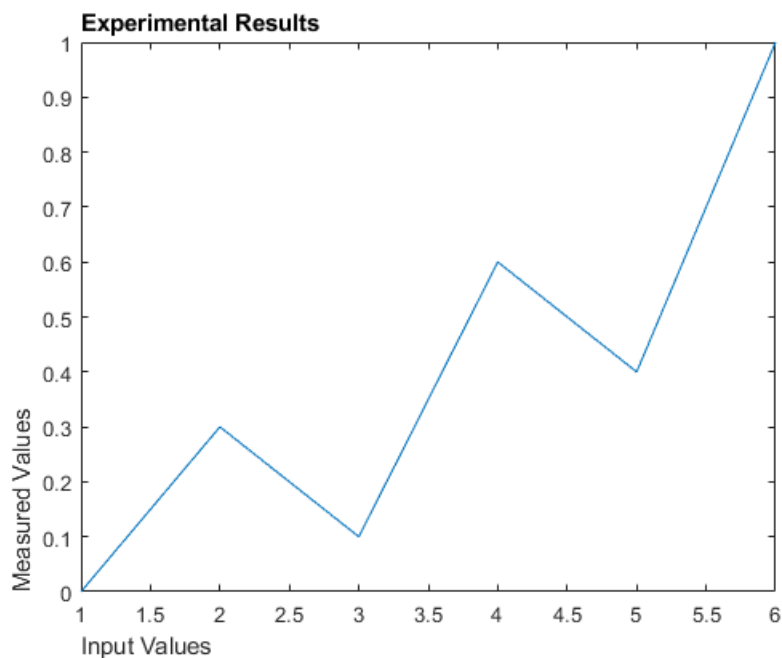


### Align Titles and Axis Labels to the Plot Box

- Align a title by setting the `TitleHorizontalAlignment` property of the axes to 'left', 'right', or 'center'.
- Align an axis label by setting the `LabelHorizontalAlignment` property of the ruler object to 'left', 'right', or 'center'. Use the `XAxis`, `YAxis`, or `ZAxis` property on the axes object to access the ruler object.

For example, this code creates a plot with a left-aligned title and axis labels.

```
plot([0 3 1 6])
title('Experimental Results')
xlabel('Input Values')
ylabel('Measured Values')
ax = gca;
ax.TitleHorizontalAlignment = 'left';
ax.XAxis.LabelHorizontalAlignment = 'left';
ax.YAxis.LabelHorizontalAlignment = 'left';
```



### Data Tips: Customize data tip content on standalone visualizations

Add or remove rows from data tips on standalone visualizations that you create from tables. You can customize data tips for these standalone visualizations:

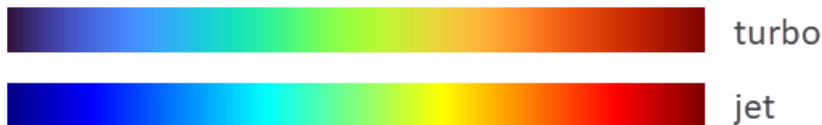
- Heatmap charts created using the `heatmap` function
- Scatter plots with marginal histograms created using the `scatterhistogram` function
- Geographic bubble charts created using the `geobubble` function
- Parallel coordinates plots created using the `parallelplot` function

To add or remove rows, right-click on the chart and hover over **Modify Data Tips**. Then, select or deselect table variables.

## turbo Colormap: jet colormap alternative with more perceptually uniform transitions

Use the `turbo` colormap to display visualizations using a colormap that is similar to `jet`, but with more perceptually uniform transitions along the color scale. Perceptually uniform colormaps help you to visualize the differences in your data more accurately.

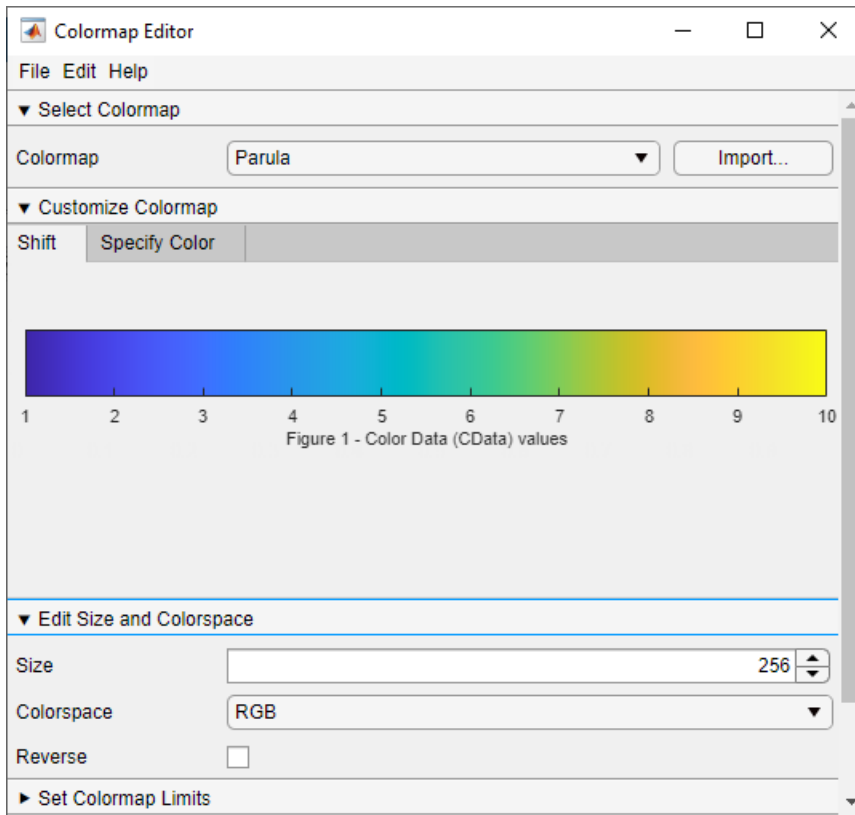
These colorbars show the difference between the `turbo` and `jet` colormaps. The `turbo` colormap is on top, and the `jet` colormap is on the bottom.



## Colormap Editor: Customize colormaps using modernized interface

Customize colormaps using the modernized interface of the Colormap Editor. Starting in R2020b, you can use the Colormap Editor to:

- Import a colormap as an  $m$ -by-3 array from the workspace.
- Save a customized colormap as a workspace variable.
- Shift the placement of colors in the colormap.
- Reverse the order of colors in the colormap.
- Change the number of elements in the colormap.



## boxchart Function: Use color to differentiate between box charts

Use color to differentiate between box charts by specifying the 'GroupByColor', cgroupdata name-value pair argument of `boxchart`. The function creates a box chart for each group of data and assigns the same color to groups with the same cgroupdata value. You can specify the color grouping variable with or without a positional grouping variable. For an example, see [Use Positional and Color Grouping Variables](#).

## im2gray and cmap2gray: Convert images and colormaps to grayscale

Use the `im2gray` function to convert colored images to grayscale. Use the `cmap2gray` function to convert a colormap to grayscale.

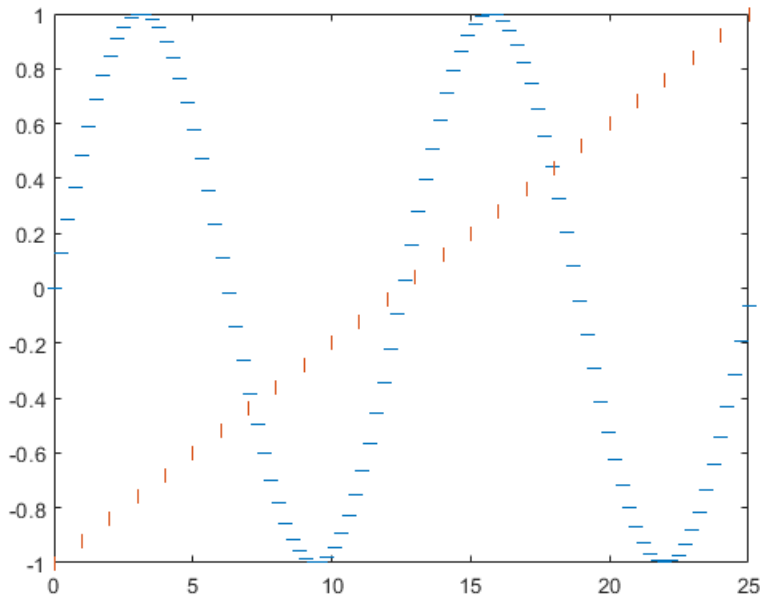
## validatecolor Function: Calculate normalized RGB triplets for color names, hexadecimal color codes, or integer values

Use the `validatecolor` function to calculate the normalized RGB triplet values for colors specified in any of the following ways:

- Color names, such as 'red' or 'green'
- Hexadecimal color codes, such as '#FF0000' or '#F00'
- Integer RGB triplets, such as `uint8([255 0 0])`

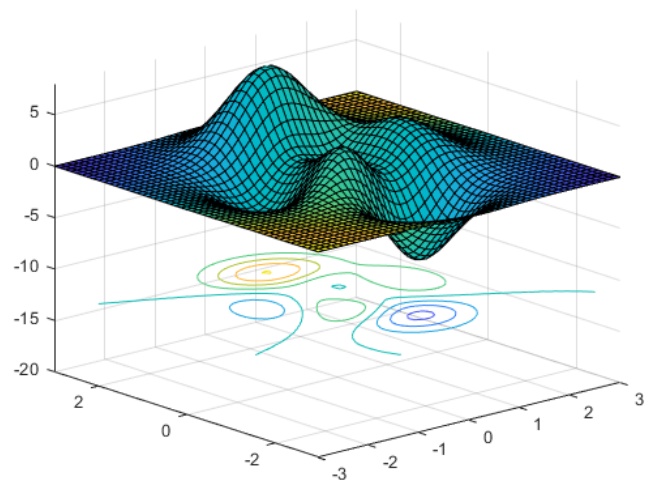
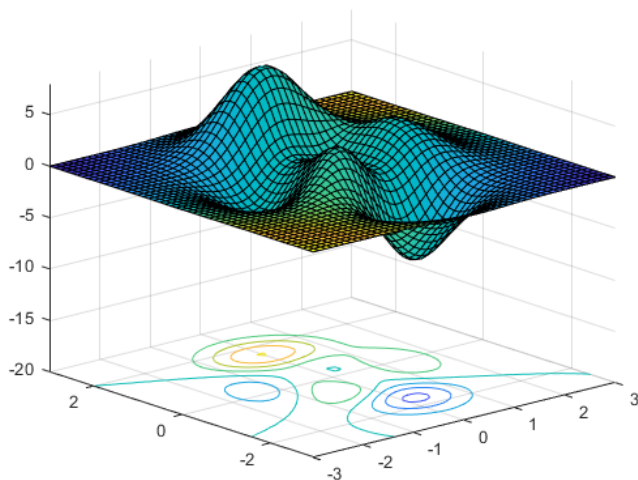
## Markers: Specify horizontal or vertical line markers for plots

Create plots with horizontal or vertical line markers when you call plotting functions such as `plot` or `scatter`. Specify horizontal line markers using the underscore ('\_') symbol. Specify vertical line markers using the pipe ('|') symbol.



## surf and mesh Functions: Specify Z-level for contours on surface and mesh plots

When you create a surface or a mesh plot with contour lines using the `surf` or `mesh` functions, you can display the contours at any z-level by specifying the `ZLocation` property of the `Contour` object. The `ZLocation` property can have a value of 'ZMin', 'ZMax', or a numeric value. The default value is 'ZMin', which corresponds to the lowest level shown in the plot box.



## animatedline Function: Create animated lines in polar plots

Create animated polar plots by calling the `animatedline` function and specifying a `PolarAxes` object as the first input argument.

## colororder Function: Control colors in geographic bubble charts

The `colororder` function now supports charts created with the `geobubble` function.

## Functionality being removed or changed

### Calling the `alpha` function with the `alphadata`, `facealpha`, or `alphadatamapping` arguments changes Scatter objects in the axes

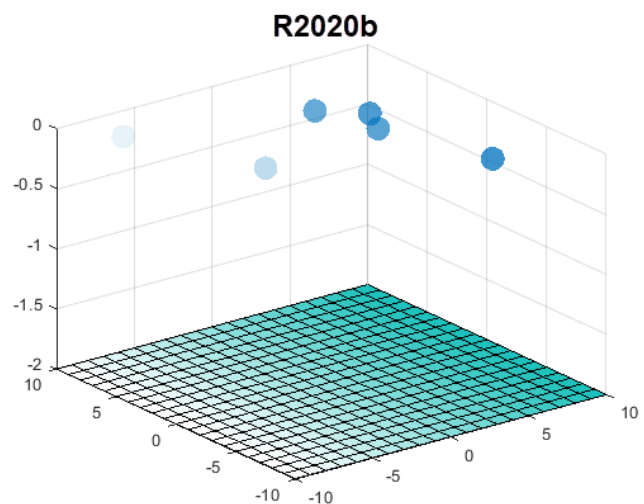
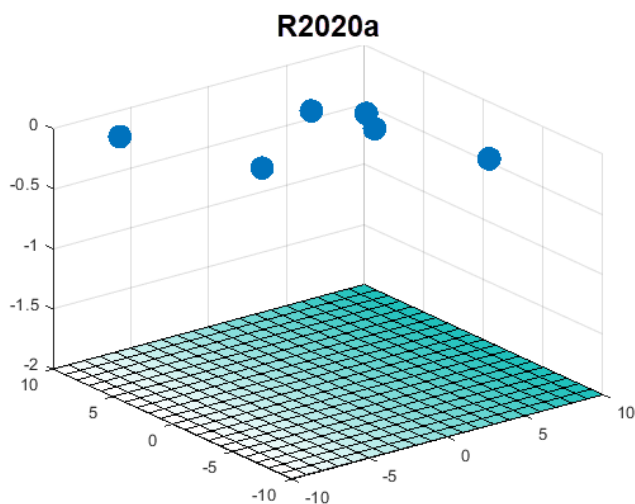
*Behavior change*

Scatter objects in the axes are affected when you call the `alpha` function with the `alphadata`, `facealpha`, or `alphadatamapping` arguments without specifying a particular object within the axes to modify.

In R2020a and earlier releases, the `alphadata`, `facealpha`, and `alphadatamapping` arguments have no effect on Scatter objects in the axes.

For example, this code creates a surface plot and a scatter plot, and then calls the `alpha` function to vary the transparency along the x-dimension. In R2020a, only the surface plot changes when you call the `alpha` function. In R2020b, both plots are affected.

```
[X,Y] = meshgrid(-10:10);
Z = ones(21,21) * -2;
surf(X,Y,Z)
hold on
scatter([-8 2 4 -5 5 3],[7 4 2 -1 -7 0],200,'filled')
alpha('x')
```



To prevent Scatter objects from changing, specify the object to modify as the first argument to the `alpha` function. For example, to update the preceding code, call the `surf` function with an output argument `s`. Then pass `s` to the `alpha` function to modify only the Surface object.

```
[X,Y] = meshgrid(-10:10);
Z = ones(21,21) * -2;
s = surf(X,Y,Z)
hold on
scatter([-8 2 4 -5 5 3],[7 4 2 -1 -7 0],200,'filled')
alpha(s,'x')
```

### **Colormap 'default' option for heatmap displays the blue colormap instead of parula**

#### *Behavior change*

Setting the colormap on a heatmap chart to 'default' sets the chart's colormap to the default blue colormap for heatmap charts. In R2020a and previous releases, the 'default' option changes the colormap to parula.

To specify the default colormap for a heatmap chart, pass the chart to the `colormap` function.

```
h = heatmap(rand(10));
colormap(h,'default')
```

Only heatmap charts are affected by this change.

### **rbbox and dragrect do not display rectangles outside of the figure window**

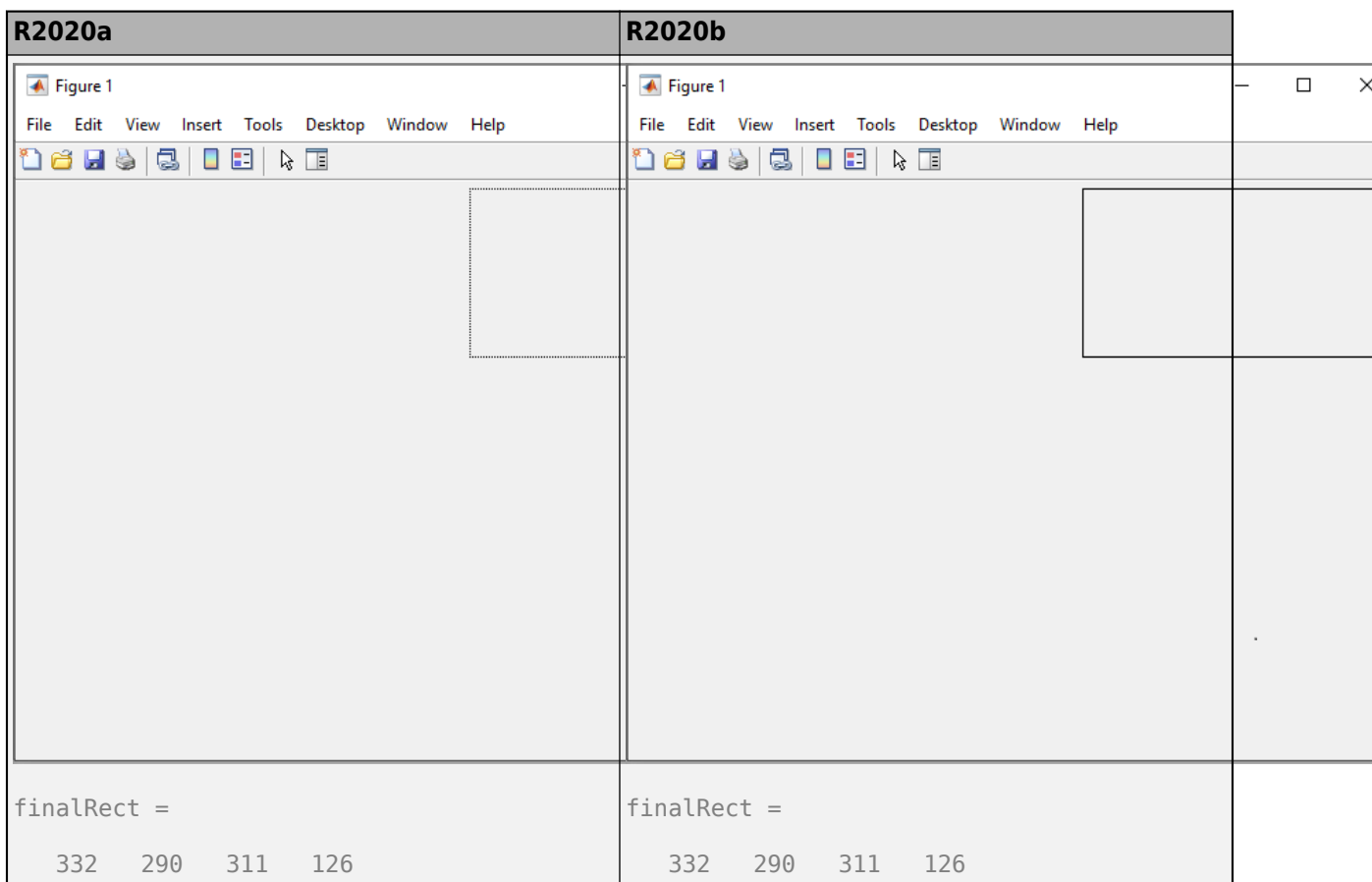
#### *Behavior change*

If you begin dragging a rectangle using the `rbbox` or `dragrect` functions and then move part of it past the edge of the figure window, only the part of the rectangle that is inside the window is visible. In R2020a and earlier releases, the whole rectangle is visible, even the part that is outside of the figure window.

This change does not affect the values returned by `rbbox` or `dragrect`.

This code and table compares the behavior of `rbbox` in R2020a and R2020b.

```
waitforbuttonpress
[finalRect] = rbbox
```



### Data tips on charts created from tables display the first dimension name and row names *Behavior change*

When you create a chart from a table, the data tips include the row information you provide. For instance, if you change the first dimension name of the table, then the data tips display that name instead of **Row**. In addition, if you specify row names for the table, then the data tips display the row names instead of the row numbers.

By default, the first dimension name of table arrays is 'Row' and the row names are empty. This means the behavior in R2020b only differs from earlier releases when you change the `DimensionNames` or `RowNames` properties of the table array.

For example, load the `patients` data set and create a table. Change the dimension names from `{'Row', 'Variables'}` to `{'Patient', 'Data'}` and the row names from `{}` to `LastName`. Then, create a scatter plot with marginal histograms. The image shows the differences in the data tips for R2020a and R2020b.

```
load patients
tbl = table(LastName, Age, Gender, Height, Weight);
tbl.Properties.DimensionNames = {'Patient', 'Data'};
tbl.Properties.RowNames = LastName;
scatterhistogram(tbl, 'Height', 'Weight')
```



The charts affected by this change are:

- Heatmap charts created using the `heatmap` function
- Scatter histogram charts created using the `scatterhistogram` function
- Geographic bubble charts created using the `geobubble` function
- Parallel coordinates plots created using the `parallelplot` function



## App Building

### **uitable Function: Configure column widths to use weighted variable or to automatically adjust to fit data**

Configure the column widths of table UI components. You can use a weighted variable width or you can automatically adjust the width to fit the data and column names.

To specify a weighted variable width, set the `ColumnWidth` property value to a number paired with an 'x' character (for example, '2x'). To configure columns to automatically adjust to column names and data, set the `ColumnWidth` property value to 'fit'.

The 'fit' and 'Nx' `ColumnWidth` property values are supported only in App Designer and `uitable`-based apps.

For more information, see [Table Properties](#).

### **scroll Function: Scroll to the top or bottom of a text area programmatically**

Use the `scroll` function to scroll to the top or bottom of a text area component programmatically.

### **WindowStyle Property: Create modal UI figures**

To restrict keyboard and mouse interactions to a specific UI figure window, set the `WindowStyle` property to 'modal'. For more information see, [UI Figure Properties](#).

### **Icon Property: Specify custom icons for UI figure windows and toolbar push and toggle tools**

You can now add custom icons to figures created with the `uitable` function. You can also specify custom icons as image files for toolbar push and toggle tools in App Designer or `uitable`-based apps. To specify a custom icon, set the `Icon` property value to an image file or an m-by-n-by-3 truecolor array.

For push and toggle tools created in App Designer and `uitable`-based apps, use the `Icon` property to specify icons instead of the `CData` property.

For more information, see [UI Figure Properties](#), [PushTool Properties](#) or [ToggleTool Properties](#).

### **WordWrap Property: Wrap long text to fit the width of certain UI components**

Use the `WordWrap` property to prevent text from getting clipped horizontally when the width of your UI component is smaller than the text you want to display. `Label`, `CheckBox`, `RadioButton`, `Button`, `StateButton`, `ToggleButton`, and `TextArea` objects support the `WordWrap` property.

Setting the `WordWrap` property value to `'on'` breaks the text into new lines so that each line fits within the component. It avoids breaking words when possible. When the value is set to `'off'` the text does not wrap.

For more information, see `Label` Properties.

## **Enable Property: Turn interaction off and on for buttons and panel groups**

To control whether a panel or button group responds to user interaction, use the `Enable` property.

For example, when the `Enable` property of a panel is set to `'on'` you can interact with it, and with UI components within it as long as they are enabled. When the `Enable` property is set to `'off'`, you cannot interact with the panel or its content. Changing the value of the `Enable` property for a panel does not change the value of the `Enable` property for UI components contained within it. The `Enable` property for button groups behaves in the same way.

The `Enable` property is supported only for panels and button groups in App Designer and `uifigure`-based apps.

For more information, see `Panel` Properties.

## **BackgroundColor Property: Set the background color for grid layouts**

Set the `BackgroundColor` property on `GridLayout` objects (created with the `uigridlayout` function). The default background color is the same as the default color for all parent containers, such as figures and panels.

## **Custom Components: Develop your own class of UI components**

Define your own class of UI components by creating a subclass of the `matlab.ui.componentcontainer.ComponentContainer` base class. Create a class to make composite UI components that use multiple MATLAB UI components or graphics objects.

Creating a class has these benefits:

- A convenient interface — When users want to customize an aspect of your UI component, they can set a property rather than having to modify and rerun your code. Users can modify properties at the command line or inspect them in the Property Inspector.
- Encapsulation — Organizing your code in this way allows you to hide implementation details from your users. You implement methods that perform calculations and manage the underlying graphics objects.

For more information, see `UI Component Development Overview`.

## **App Designer: Allow only one running instance of your app at a time**

In App Designer, you can select whether your app can run multiple instances at a time or only a single instance.


To change the run behavior of your app, select the App node from the **Component Browser**. Then, from the **Code Options** section of the **Inspector** tab, select or clear **Single Running Instance**.

When **Single Running Instance** is selected and you run the app multiple times, MATLAB reuses the existing instance and brings it to the front rather than creating a new one. When **Single Running Instance** is not selected, then a new app instance is created each time you run it, while existing instances of the app also continue to run.

Selecting or clearing **Single Running Instance** does not change the App Designer run behavior of the app. Regardless of the **Single Running Instance** option, App Designer always closes the existing app instance before creating a new one.

For more information, see [Manage Code in App Designer Code View](#).

## App Designer: Change the stacking order of UI components

To change the stacking order (also known as *z-order*) of components in your app, select a component. Then, in the App Designer toolstrip, expand the **Reorder**  drop-down menu from the **Arrange** section of the **Canvas** tab and select a reorder option. Alternatively, right-click a component and select an option from the **Reorder** menu.

## App Designer: Add and configure toolbar components on the App Designer canvas

Add a toolbar component to your app by dragging one from the **Component Library** onto the canvas. Configure the toolbar by adding push or toggle tools to it. Then, add icons to the tools and configure their callbacks.




## App Designer: Draw UI components on the App Designer canvas

In App Designer, you can now draw UI components from the **Component Library** on the canvas.

To draw a component on the canvas, select it from the **Component Library** and then move your cursor over the canvas. The cursor changes to a crosshair. Click your mouse to add the component to the canvas in its default size, or click and drag to size the component as you add it.

## App Designer: Find differences and merge apps

You can use the Comparison Tool to find differences between apps. The Comparison Tool highlights differences in the code of two apps. To start a comparison of two apps, go to the **Home** tab, and in the **File** section, click **Compare**, and then select the files that you want to compare. To start a comparison from the Current Folder browser, select a file, right-click, and select **Compare Against**.

You also can merge changes in callback and utility functions from one file to the other. Merging changes can be useful when resolving conflicts between different versions of an app. To merge two apps, in the Comparison Tool, select the  **Merge Mode** button to start the merge. Use the  button to replace content in the right pane with content from the left pane. The right pane contains the merged result. To save the result, click the  **Save Result** button.

For more information, see [Compare and Merge Apps](#).

## Graphics Support: Create more plots in apps with full support for any type of axes

All plotting workflows are now supported in App Designer apps and in apps you create programmatically with the `uifigure` function. For example:

- You can add more types of plots to an app, including `pareto`, `plotmatrix`, and `boxplot`.
- You can place any type of axes or standalone visualization (such as a heatmap) directly into a scrollable container or in a `GridLayout`.
- You can place a tiled chart layout directly into a scrollable container or in a `GridLayout`.

Also, `UIAxes` objects now have the same properties and options as `Axes` objects have for customizing their appearance and behavior:

- You can use the `InnerPosition` property to align UI components with the plot box.
- You can use the `PositionConstraint` property to control the space around the plot box when you add or modify decorations such as titles and axis labels.
- The `Units` property of a `UIAxes` object now supports all units: `'pixels'`, `'normalized'`, `'inches'`, `'centimeters'`, `'points'`, and `'characters'`. The default value is `'pixels'`.
- You can use the `ButtonDownFcn`, `PickableParts`, and `HitTest` properties to make the `UIAxes` object capture and respond to mouse clicks.

## Graphics Support: Identify coordinates and display text by clicking or tapping

You can now identify coordinates or display text at specified locations by clicking or tapping within App Designer figures or figures created with the `uifigure` function. To identify axes coordinates, call the `ginput` function. Then, select the locations to identify by clicking or tapping. To display text, call the `gtext` function and specify the text as an argument. Then, select the locations for the text by clicking or tapping. For more information about using the `ginput` and `gtext` functions within App Designer, see [Display Graphics in App Designer](#).

Additionally, you can now migrate GUIDE apps that use `ginput` or `gtext`. For more information, see [GUIDE Migration Strategies](#).

## App Capture: Capture user interfaces using `exportapp` and `getframe`

Capture the all the graphical content in an app, including UI components, as an image file, a PDF file, or an image array in your MATLAB workspace.

- Call the `exportapp` function to capture the content as a JPEG, PNG, or TIFF file, or as a PDF file containing vector graphics.
- Call the `getframe` function with a return argument to capture the content as an image array in your workspace. To ensure all the UI components are included, specify the figure as the first input argument of the `getframe` function. The figure can be one created with the `uifigure` function, or it can be the figure in an App Designer app. By default, App Designer stores the figure in the `UIFigure` property of the app.

## App Testing Framework: Perform choose gestures on context menu items

The `chooseContextMenu` method enables you to test choose gestures on context menu items for UI components. For example, create a context menu with two menu items in a UI figure and choose one of the menu items.

```
fig = uifigure;
cm = uicontextmenu(fig);
m1 = uimenu(cm,'Text','Menu1');
m2 = uimenu(cm,'Text','Menu2');
fig.ContextMenu = cm;

tc = matlab.uitest.TestCase.forInteractiveUse;
tc.chooseContextMenu(fig,m1)
```

## App Testing Framework: Perform drag gestures on axes and UI axes

The app testing framework supports drag gestures on axes and UI axes. When you test a drag gesture on an axes or UI axes, the framework mimics a user manipulating the component and adjusts the axes limits based on the difference between the start and stop values. For example, create an axes with a plot and then drag from the point (3, 2) to the point (4, 2).

```
f = uifigure;
ax = axes(f);
plot(ax,1:10)

tc = matlab.uitest.TestCase.forInteractiveUse;
tc.drag(ax,[3 2],[4 2])
```

## App Testing Framework: Perform gestures on push tools and toggle tools

The app testing framework supports gestures on more UI components.

- Perform press gestures in tests on push tools.
- Perform press and choose gestures in tests on toggle tools.

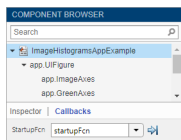
## Functionality Being Removed or Changed

### App Designer `StartupFcn` is now a callback of the app instead of the figure

*Behavior change*

The **StartupFcn** callback is now a callback of the app instead of the UI figure. Also, the app is now the root node in the App Designer **Component Browser** hierarchy. The name of the app node is the same as the name of your MLAPP file.

When you load apps created with MATLAB R2020a or earlier into R2020b, the app node is automatically added to the hierarchy. To access the **StartupFcn** callback, select the app node from the **Component Browser**. Then, select the callback from the **Callbacks** tab. The code in your `startupFcn` callback function is not changed.



**App Designer Component Browser hierarchy now uses the z-order of components***Behavior change*

The App Designer **Component Browser** hierarchy now lists components according to their stacking order (z-order) instead of their creation order.

**GridLayout background is no longer transparent***Behavior change*

GridLayout objects now have a BackgroundColor property and they are no longer transparent. The default background color is the default color for all containers (for example, figures and panels).

If your app has a grid layout in a container that has a nondefault color, then set the BackgroundColor property of the GridLayout object to that color to preserve the appearance of your app.

If your app has objects behind the grid that you want to remain visible, move those objects into the grid by making them children of the GridLayout object.

**BackgroundColor property of UIAxes has no effect***Behavior change*

Setting the BackgroundColor property on a UIAxes object no longer has any effect. The area around the plot box is transparent regardless of the value of the BackgroundColor property.

To produce the same effect as setting the background color in previous releases, create a panel with the desired BackgroundColor value, and then place the UIAxes in the panel.

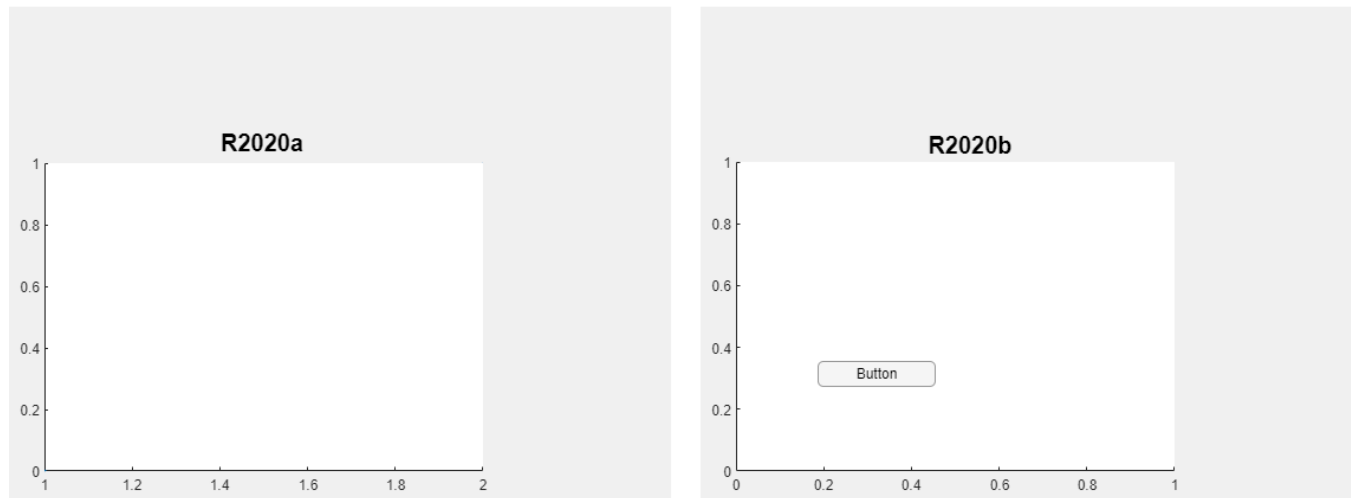
**UIAxes appear behind all other objects in the figure***Behavior change*

The stacking order (also called the z-order) of objects in the figure has changed so that UIAxes objects and their contents appear behind other UI components in the figure. This behavior is consistent with the behavior of other types of axes.

For example, this code creates a figure, a button, and then a UIAxes object.

```
fig = uifigure;  
b = uibutton(fig);  
uax = uiaxes(fig);
```

In R2020a, executing the preceding code displays the UIAxes in front of the button, as shown in the figure on the left. The figure on the right shows the behavior in R2020b, where the UIAxes appears behind UI components regardless of the order of creation.



The order of the objects listed in the `Children` property of the figure also reflects this change. The `UIAxes` object is always after UI components in the list.

```
fig.Children
```

```
ans =
```

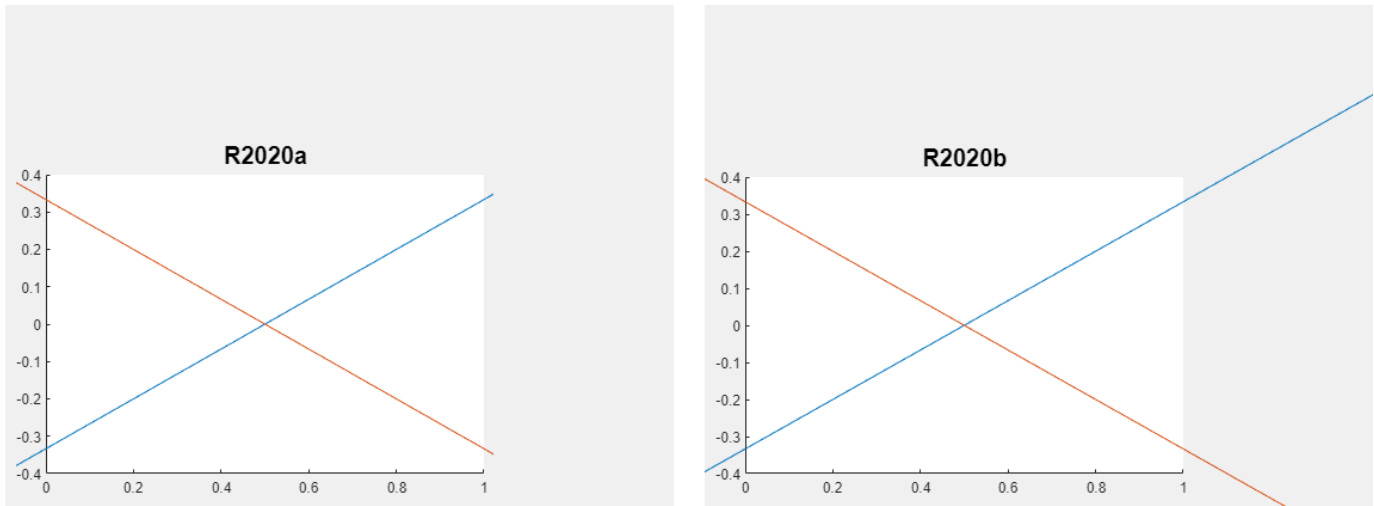
```
2×1 graphics array:
```

```
Button    (Button)
UIAxes
```

### Plots in UIAxes might extend outside the bounds of the axes

#### *Behavior change*

Plot objects such as lines might not clip to the bounds defined by the `OuterPosition` property of the `UIAxes`. A line extends beyond the bounds when its `Clipping` property is set to `'off'`. In previous releases, lines clip to the `OuterPosition` regardless of the value of the `Clipping` property. For example, the plot on the left shows the R2020a behavior, and the plot on the right shows the R2020b behavior. In both cases, the `Clipping` properties of the lines are set to `'off'`.



To prevent the axes content from overlapping with components in your app, set the `Clipping` property of each object in the axes to `'on'`.

### **Colorbars and legends displayed with UIAxes have the same parent as UIAxes**

#### *Behavior change*

When you create a plot in a `UIAxes` object, and then create a colorbar or legend for that plot, the parent object of the colorbar or legend is the same as the parent object of the `UIAxes` object. In previous releases, the parent object of the colorbar or legend is the `UIAxes` object.

### **UIAxes SizeChangedFcn callback has been removed**

#### *Errors*

The `SizeChangedFcn` callback for `UIAxes` objects has been removed. If your app requires a callback that executes when the size of the axes changes, create a `SizeChangedFcn` callback for the parent figure or another container.



## Performance

### sum Function: Improved performance summing the first dimension of numeric arrays

The performance of the `sum` function has improved when operating on the first dimension of numeric inputs:

- `sum(A,1)` and `sum(A,'all')` are faster for matrix or N-D array `A`.
- `sum(v)` is faster for row or column vector `v`.

This performance improvement was added for single inputs in R2017b, and it has now been added for all other numeric types in R2020b: `double`, `logical`, and integer data types (`int8`, `int16`, ...).

For example, if you sum the elements of a vector with  $1e9$  elements in a loop, performance in R2020b is about 1.4x faster than in R2020a.

```
function timingSum
x = 1/100;
v = x*ones(1e9,1);

tic
for i = 1:10
    b = sum(v);
end
toc
end
```

The approximate execution times are:

**R2020a:** 2.0 s

**R2020b:** 1.4 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSum`.

### Version History

This performance improvement arises from code changes that also lead to a different set of round-off behaviors in `sum`. The new algorithm reduces the amount of round-off error in calculations, which leads to a more accurate result in general. Therefore, the output of `sum` might change slightly in R2020b compared to R2020a when operating on numeric inputs, even though the results between the two versions are numerically equivalent.

### polyfit Function: Improved performance fitting data

The performance of the `polyfit` function has improved for input data of all sizes and fits of any order.

For example, if you fit a quadratic polynomial to a 500-element vector in a loop, performance in R2020b is about 25x faster than in R2020a.

```
function timingPolyfit
rng default
x = 1:500;
y = -0.3*x + 2*randn(1,500);

for i = 1:1e4
    p = polyfit(x,y,2);
end
end
```

The approximate execution times are:

**R2020a:** 3.5 s

**R2020b:** 0.14 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingPolyfit)
```

## Version History

The performance improvement arises from changes to the `polyfit` algorithm that also produce more accurate results. Therefore, the output of `polyfit` might change slightly in R2020b compared to R2020a, even though the results between the two versions are numerically equivalent.

## accumarray Function: Improved performance with fill values and certain function handles

The performance of the `accumarray` function has improved in these cases:

- When you use the default `@sum` accumulation function and specify the `fillval` input.
- When you specify the accumulation function as `@min` or `@max`.

For example, when you accumulate a vector with 10,000 elements into an output vector with 100 elements and specify a fill value, performance in R2020b is about 14x faster than in R2020a.

```
function timingAccumarrayFillVal
rng default
subs = randi(1e2,1e4,1);
val = randn(1e4,1);
sz = [1e2 1];
fillval = 1;

for k = 1:2e4
    A = accumarray(subs,val,sz,[],fillval);
end
end
```

The approximate execution times are:

**R2020a:** 11.2 s

**R2020b:** 0.8 s

Also, when you accumulate a vector with 10,000 elements into an output vector with 100 elements and specify the accumulation function as `@min` or `@max`, performance in R2020b is about 8.4x faster than in R2020a.

```
function timingAccumarrayFcnHandle
    rng default
    subs = randi(1e2,1e4,1);
    val = randn(1e4,1);
    sz = [1e2 1];

    for k = 1:2e4
        A = accumarray(subs,val,sz,@max);
    end
end
```

The approximate execution times are:

**R2020a:** 7.6 s

**R2020b:** 0.9 s

In each case, the code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingAccumarrayFillVal)
timeit(@timingAccumarrayFcnHandle)
```

## spdiags Function: Improved performance constructing sparse banded matrices

The performance of the `spdiags` function has improved by a factor of up to 3x when constructing sparse banded matrices.

For example, if you create a large banded sparse matrix of order  $1e7$  with 11 nonzero diagonals, performance in R2020b is about 2.6x faster than in R2020a.

```
function timingSpdiags
    n = 1e7;
    B = randn(n,11);
    d = -5:5;

    tic
    A = spdiags(B, d, n, n);
    toc
end
```

The approximate execution times are:

**R2020a:** 11.1 s

**R2020b:** 4.2 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSpdiags`.

## uicontrol: Improved performance when setting multiple items in a list box

Setting property values on a `uicontrol` shows improved performance when operating on a large number of selected items. For example, this code is about 117x faster than in the previous release:

```
function timingTest
lb = uicontrol;
lb.Items = "Item " + (1:1000);
lb.ItemsData = 1:1000;
lb.Multiselect = true;
lb.Value = 1:800;
end
```

The approximate execution times are:

**R2020a:** 4.7 s

**R2020b:** 0.04 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingTest)
```

## uitree: Improved performance when creating many nodes in a tree

The `uitreenode` function shows improved performance when creating a large number of nodes parented to a `uitree`. For example, this code that creates a tree with 3,000 nodes is about 2.4x faster than in the previous release:

```
function timingTest
t1to3000 = [];
fig = uifigure;
tree = uitree(fig);
drawnow
for i=1:3000
    treeNodeText = ['t' num2str(i)];
    t1to3000(i) = uitreenode('Parent',tree,'Text',treeNodeText);
end
drawnow
end
```

The approximate execution times are:

**R2020a:** 81.4 s

**R2020b:** 33.7 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

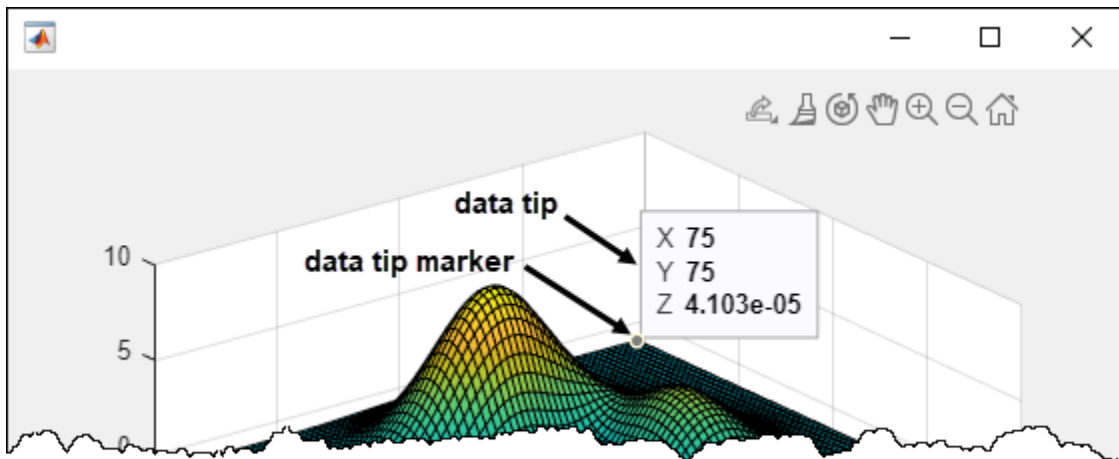
```
timeit(@timingTest)
```

## Data Tip Markers: Improved rendering performance of data tip markers in surface plots of large data sets created in UI figures and MATLAB Online

In figures created with the `uifigure` function and in MATLAB Online, data tip markers for surface plots of large data sets render faster and move more continuously in R2020b than in R2020a. This improvement can be seen when the axes are created with either the `axes` or `uiaxes` function.

For example, on a *Windows 10, Intel Xeon W-2133 CPU @ 3.6 GHz* test system with an *NVIDIA® Quadro P600* graphics card, when you move your mouse quickly over the plot lines created by this code, the data tip marker moves more smoothly and tracks your mouse motion more closely in R2020b than in R2020a.

```
fig = uifigure;  
ax = axes(fig);  
surf(ax,peaks(75))
```



## Software Development Tools

### **Code Compatibility Report: Unsupported Functionality Will Now Issue Warning**

The code compatibility report will now check and notify you of some unsupported functionality within the code.

### **Dependency Analyzer: Export to archive and generate a dependency report**

Dependency analysis for projects is improved and includes new capabilities:

- Export files in the dependency graph to an archive.
- Save the dependency analysis results in a printable report.
- Investigate where in your files the dependency to a product is introduced.
- Determine which files, if any, use functionalities shared among different MathWorks products.

For more information, see [Analyze Project Dependencies](#).

### **Source Control: Improved workflow to set up Git source control**

The workflow to set up Git is improved and simplified.

- You do not need to install command-line Git to fully use Git with MATLAB. You can now merge branches using the built-in Git integration.
- For newly created projects using Git and for projects that switched to Git from another source control, MATLAB automatically generates a `.gitattributes` file and populates it with a list of common binary files. You do not need to manually register binary files to prevent corruption.

For more information, see [Set Up Git Source Control](#).

### **Projects: Change project definition file type and preserve source control history**

You can now use `matlab.project.convertDefinitionFiles` to programmatically change the project definition file management from the type selected when you first created the project to a new type. `matlab.project.convertDefinitionFiles` preserves the source control history of your project. To avoid any merging issues, make sure to convert the definition file type only once for your project.

### **Unit Testing Framework: Run tests in parallel on clusters and clouds**

You can now run your tests on clusters and clouds using MATLAB Parallel Server™. To run tests on a remote parallel pool, use the same API the framework offers for running tests on a local parallel pool. In other words, use the `runInParallel` method or the `runtests` function with the `'UseParallel'` name-value pair argument.

## Unit Testing Framework: Run tests in parallel with standalone applications

Starting in R2020b, you can create standalone applications that support running tests in parallel. To ensure that MATLAB Compiler can locate and package all of the components required for running tests in parallel, use this directive in your program:

```
##function parallel.Pool
```

For more information, see [Compile MATLAB Unit Tests](#).

## Unit Testing Framework: Report the validity of shared test fixtures

Starting in R2020b, you can report the validity of shared test fixtures to the testing framework. A shared test fixture is valid if the test environment state, configured by the fixture, is maintained throughout the test session.

To report the validity of a shared fixture, override the `needsReset` method of the `matlab.unittest.fixtures.Fixture` class. When running multiple test classes as a single suite, the framework tears down a shared test fixture and sets it up for the subsequent classes if `needsReset` returns `true`.

## App Testing Framework: Perform choose gestures on context menu items

The `chooseContextMenu` method enables you to test choose gestures on context menu items for UI components. For example, create a context menu with two menu items in a UI figure and choose one of the menu items.

```
fig = uifigure;
cm = uicontextmenu(fig);
m1 = uimenu(cm,'Text','Menu1');
m2 = uimenu(cm,'Text','Menu2');
fig.ContextMenu = cm;

tc = matlab.unittest.TestCase.forInteractiveUse;
tc.chooseContextMenu(fig,m1)
```

## App Testing Framework: Perform drag gestures on axes and UI axes

The app testing framework supports drag gestures on axes and UI axes. When you test a drag gesture on an axes or UI axes, the framework mimics a user manipulating the component and adjusts the axes limits based on the difference between the start and stop values. For example, create an axes with a plot and then drag from the point (3, 2) to the point (4, 2).

```
f = uifigure;
ax = axes(f);
plot(ax,1:10)

tc = matlab.unittest.TestCase.forInteractiveUse;
tc.drag(ax,[3 2],[4 2])
```

## App Testing Framework: Perform gestures on push tools and toggle tools

The app testing framework supports gestures on more UI components.

- Perform press gestures in tests on push tools.
- Perform press and choose gestures in tests on toggle tools.

## Functionality being removed or changed

### SVN cleanup no longer removes unversioned or ignored files

#### *Behavior change*

Starting in R2020a Update 5, SVN cleanup only removes stale locks and unfinished transactions. It does not remove unversioned or ignored files.

You can manually remove unversioned and ignored files.

- 1 In the Current Folder browser, click the **SVN** header to sort files by their SVN status.
- 2 Select the **Not Under Source Control** files.
- 3 Right-click and select **Delete**.

For more information, see [Get SVN File Locks](#).

### ProfileReport has been removed

#### *Errors*

The ProfileReport class has been removed. Use CoverageReport or CoberturaFormat instead. Compared to ProfileReport, these two classes generate more accurate code coverage reports.

To update your code, change instances of ProfileReport to CoverageReport or CoberturaFormat. This table shows an example of how you can update your code.

Before	After
<pre>import matlab.unittest.TestRunner import matlab.unittest.plugins.CodeCoverage import matlab.unittest.plugins.codecoverage  runner = TestRunner.withNoPlugins; plugin = CodeCoveragePlugin.forFolder('myTests',     'Producing',ProfileReport); runner.addPlugin(plugin)</pre>	<pre>import matlab.unittest.TestRunner import matlab.unittest.plugins.CodeCoveragePlugin import matlab.unittest.plugins.codecoverage.CoverageReport  runner = TestRunner.withNoPlugins; plugin = CodeCoveragePlugin.forFolder('myTests', ...     'Producing',CoverageReport); runner.addPlugin(plugin)</pre>

To create a MATLAB Profiler Coverage Report without specifying a ProfileReport format, see [Determine Code Coverage Using the Profiler](#).

### Indexing order has changed in choose gestures within button groups

#### *Behavior change*

With the app testing framework, you can choose a radio button or toggle button using the button label or its index inside the button group. Starting in R2020b, when you choose a radio button or toggle button using an index, the framework indexes into the Buttons property of the ButtonGroup object. In previous releases, the framework indexes into the Children property of the ButtonGroup object. With this change, the index corresponds to the order in which buttons are created. For example, create a button group that has six toggle buttons.

```
f = uifigure;
bg = uibuttongroup(f);
```



```

tb1 = uitogglebutton(bg, 'Position', [11 165 140 22], 'Text', 'One');
tb2 = uitogglebutton(bg, 'Position', [11 140 140 22], 'Text', 'Two');
tb3 = uitogglebutton(bg, 'Position', [11 115 140 22], 'Text', 'Three');
tb4 = uitogglebutton(bg, 'Position', [11 90 140 22], 'Text', 'Four');
tb5 = uitogglebutton(bg, 'Position', [11 65 140 22], 'Text', 'Five');
tb6 = uitogglebutton(bg, 'Position', [11 40 140 22], 'Text', 'Six');

```

This table shows the outcome of the choose gesture on a toggle button that is specified with index 2.

Test	Starting in R2020b	R2020a and Earlier
<pre> tc = matlab.uitest.TestCase; tc.choose(bg,2) </pre>	<p>MATLAB chooses toggle button <code>tb2</code>, which is the second element of the <code>bg.Buttons</code> array. The <code>tc.choose(bg,2)</code> syntax is equivalent to <code>tc.choose(bg, 'Two')</code>.</p>	<p>MATLAB chooses toggle button <code>tb5</code>, which is the second element of the <code>bg.Children</code> array. The <code>tc.choose(bg,2)</code> syntax is equivalent to <code>tc.choose(bg, 'Five')</code>.</p>

If this behavior change impacts your code, update the index or replace it with the appropriate button label.

## External Language Interfaces

### **C++ Interface: Support for nullptr**

You can now pass `nullptr` to C++ functions. For more information, see `nullptr` Argument Types.

### **C++ Interface: Create interface with C++ source files**

You can build a MATLAB interface to a C++ library or algorithm from source files that contain complete implementations for the library. Use the `SupportingSourceFiles` name-value pair in the `clibgen.generateLibraryDefinition` function. For an example, see `Publish Interface to C++ Library Using Source Files`.

### **Python: Version 3.8 support**

MATLAB now supports CPython 3.8, in addition to existing support for 2.7, 3.6, and 3.7. For more information, see `Configure Your System to Use Python`.

### **Python: Terminate Python interpreter and start new one in same MATLAB session**

When you run the Python interpreter in out-of-process mode, you can terminate the interpreter and start a new one without restarting MATLAB. For more information, see `Reload Out-of-Process Python Interpreter`.

### **mxCreateString C Matrix API functions: UTF-8 support**

The C Matrix API `mxCreateString` function now accepts UTF-8 encoded data in addition to supporting local code page (LCP) encoded strings for backwards compatibility. The `mxArrayToUTF8String` function returns UTF-8 encoded data.

Also, the `mexPrintf`, `mexEvalString`, `mexEvalStringWithTrap`, `mexErrMsgIdAndTxt`, and `mexWarnMsgIdAndTxt` C functions now accept UTF-8 encoded data in addition to supporting LCP encoded strings for backwards compatibility.

### **MATLAB Data API: Create `matlab::data::Object` arrays**

Use the `matlab::data::ArrayFactory` `createArray` method to create a `matlab::data::ObjectArray` and the `createScalar` method to create a scalar `matlab::data::ObjectArray`. You can use `[]` indexing and a `matlab::data::TypedIterator<T>` to access the elements of a `matlab::data::ObjectArray`.

## Compiler support changed for building C++ interfaces, MEX files, and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	gcc versions 5.x and higher. If you use version 4.0 or earlier, then MATLAB displays a warning.	Linux

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

## Functionality being removed or changed

### Java packages to be removed

*Behavior change in future release*

Certain Java libraries (JAR files) used by MathWorks products might not be included in future releases. To continue using functionality from these libraries, install the JAR files and add them to the class path used by MATLAB. For information about adding JAR files to the static class path, see Static Path.

### Undocumented com.mathworks Java packages to be removed

*Warns*

The MATLAB Code Compatibility Report lists com.mathworks Java classes as **Unsupported functionality that might cause errors**. Except for documented interfaces and classes in the com.mathworks.engine and com.mathworks.matlab.types packages (the MATLAB Engine API for Java) and the Java Client API for use with MATLAB Compiler SDK™ and MATLAB Production Server™, all other com.mathworks Java interfaces and classes are undocumented and might be modified or removed without warning in future releases. For code stability, avoid using any of these undocumented Java interfaces or classes.

## Hardware Support

### **Live Editor Task: Interactively capture images from USB Webcam interactively and generate MATLAB code in a live script.**

Use the **Acquire Webcam Image** Live Editor Task in the MATLAB Support Package for USB Webcams, to connect to a webcam, set properties, and capture images without writing MATLAB code. The task provides you with controls that help you set webcam resolution values and other image-specific and device-specific properties. The task also automatically generates MATLAB 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, see [Add Interactive Tasks to a Live Script \(MATLAB\)](#).

### **Functionality being removed or changed**

#### **mkrMotorCarrier function will be removed in a future release**

*Warns*

Starting R2020b, using the `mkrMotorCarrier` function will throw a warning message. This function will be removed in a future release. Use the `motorCarrier` instead of `mkrMotorCarrier` to connect to the MKR Motor Carrier hardware.

#### **MKRMotorCarrier library will be removed in a future release**

*Warns*

Starting R2020b, using the `'MKRMotorCarrier'` library will throw a warning message. This library will be removed in a future release. Use the `'MotorCarrier'` library instead of `'MKRMotorCarrier'` when creating a connection to the Arduino, hardware using the `arduino` object.

# R2020a

---

**Version: 9.8**

**New Features**

**Bug Fixes**

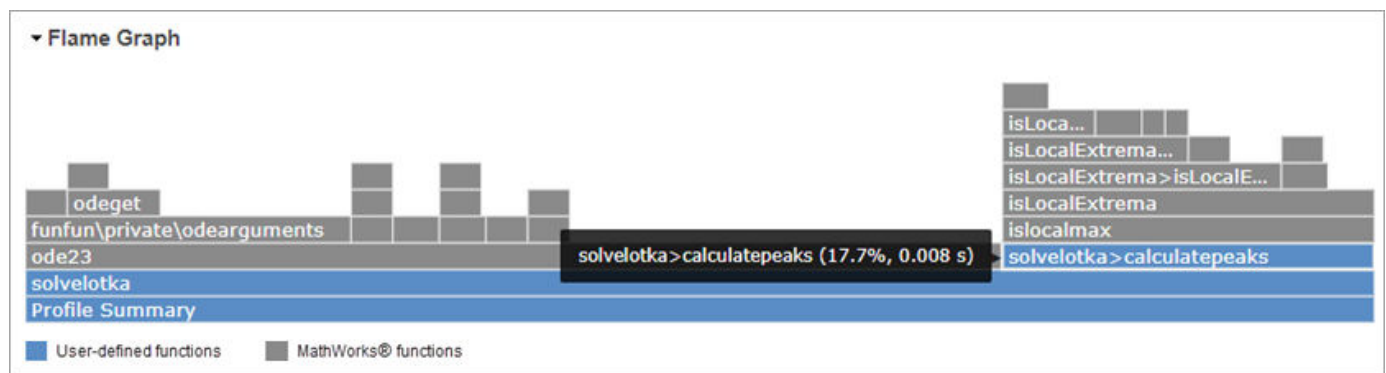
**Version History**

## Environment

### Profiler Flame Graphs: Investigate and improve the performance of your code visually

The redesigned Profiler now includes a flame graph that allows you to visually explore the execution performance results of your code. You can use the flame graph to identify functions that use a significant amount of time.

For example, this flame graph shows the performance results for a function called `solvelotka`. The wide graphs represent the functions that use the most time. You can use the Profiler to explore those functions and determine whether execution can be sped up.



For more information, see [Profile Your Code to Improve Performance](#).

### Live Editor Loop Execution: Improved performance when running loops in live scripts

Loops run significantly faster in live scripts. For example, a live script containing this code, which runs a loop one million times and displays the current loop iteration every ten thousand iterations, is approximately 42x faster.

```
for t = 1:1000000
    if ~mod(t, 10000)
        disp(t)
    end
end
```

The approximate execution times are:

**R2019b:** 2.291 s

**R2020a:** 0.054 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

The more iterations a loop contains, the greater the performance improvement becomes.

## Live Editor Animation Output: Improved performance when animating plots in live scripts

For-loop animations display faster in live scripts. For example, a live script containing this code, which creates a for-loop animation of a sine wave plot, is approximately 1.3x faster.

```
tic
x = 0:0.1:10*pi;
y = sin(x);
xlim([0 10*pi])
ylim([-1 1])
hold on
p = plot(x(1),y(1));
for k=1:length(x)
    p.XData = x(1:k);
    p.YData = y(1:k);
    drawnow
end
toc
```

The approximate execution times are:

**R2019b:** 8.875 s

**R2020a:** 6.633 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

## Live Editor Responsiveness: Improved performance with extended use

The Live Editor maintains its interaction performance (such as typing and scrolling) when running MATLAB over extended periods of time. In previous releases, the interaction performance of the Live Editor decreased over time.

## Live Editor Control Value Changes: Run all necessary code on value changes

You can configure a control to run the current section and any stale code above it when the control value changes. This ensures that when the value of the control changes, any modified or not yet run sections above are run as well. To configure the control, right-click the control and select **Configure Control**. Then, in the **Run** field, select the **Current section and modified or not yet run sections above** option.

For more information about adding controls to a live script, see [Add Interactive Controls to a Live Script](#).

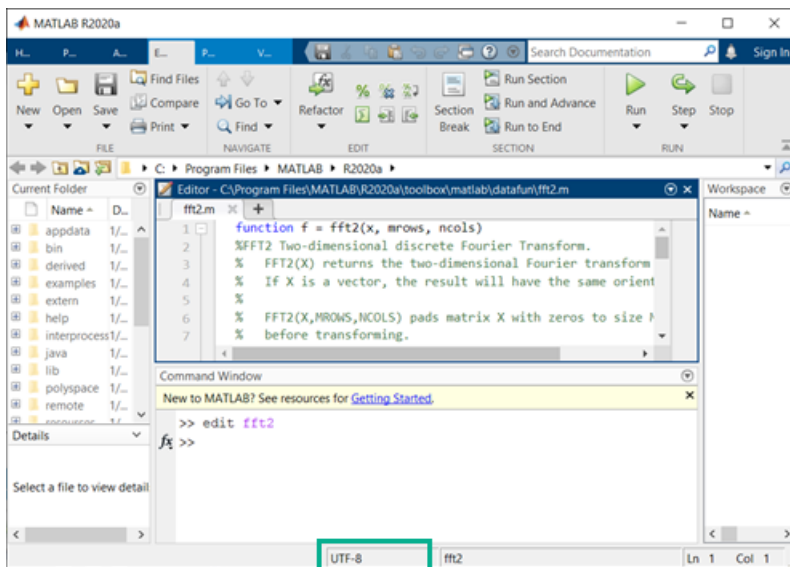
## File Encoding: Save MATLAB code files (.m) and other plain text files as UTF-8 encoded files by default

As of R2020a, MATLAB defaults to saving new plain text files using UTF-8 without a byte-order-mark (BOM). This includes files created with the MATLAB Editor and the `edit` or `fopen` functions, as well

as MATLAB log files and files created with the `diary` function. In the Editor, this includes all MATLAB code files with a `.m` extension, such as scripts and functions. When opening existing files, the Editor and other functions like `type` or `fopen` automatically determine the current encoding. The Editor saves files with their current encoding unless a different one is selected from the **Save As** dialog. For example, to save a file using the legacy locale-specific encoding for compatibility with an earlier release of MATLAB, select **Save > Save As...** in the **File** section on the **Editor** tab. In the dialog box that appears, select the desired encoding from one of the **Save as type** options.

MATLAB uses Unicode internally so that it can represent all letters and symbols, regardless of platform, language, or locale. UTF-8 was adopted as MATLAB's default character encoding to ensure that all Unicode code points can be correctly represented in files and byte streams. MATLAB also supports other character encodings for backwards compatibility and interoperability. For more information, see [Locale Setting Concepts for Internationalization](#).

The current encoding is displayed next to the file name in the Editor status bar or, if the Editor Window is docked, the Desktop status bar.



## Multiple Sources in Help Browser: Search MathWorks documentation and custom documentation together in a single browser

When you search the documentation in the Help browser, the Help browser displays both MathWorks documentation results and installed custom documentation results. To switch between the two types of results, use the **Source** facets that appear on the left side of the page. For example, to view the MathWorks documentation results, select **MathWorks** as the source. To view the installed custom documentation results, select **Supplemental Software** as the source.

## Web Documentation: View MathWorks documentation on the web without logging in

If your documentation preferences are set to view documentation on the web, you now can view the documentation for most products without logging in.

For more information about setting your documentation location, see [Help Preferences](#).



## **Internationalization: UTF-8 as system encoding on Mac and Windows platforms**

On the Mac platform, MATLAB uses UTF-8 as its system encoding to align with macOS.

On the Windows platform, if the **Use Unicode UTF-8 for worldwide language support** option is enabled in the Windows **Region** settings dialog box, then MATLAB uses UTF-8 as its system encoding.

## Language and Programming

### **switch Function: Compare objects more flexibly**

MATLAB enables you to use objects of a class in `switch` statements if the objects support the `eq` function. In previous releases, you can use objects of a class in `switch` statements only if the output of the overloaded `eq` function is a logical value. Starting in R2020a, the output of `eq` can be either a logical value or convertible to a logical value. For more information, see [Objects In Conditional Statements](#).

### **copyfile and movefile Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage**

You can now use the `copyfile` and `movefile` functions to work with remote files and folders. To access remote locations, you must specify the full path using a uniform resource locator (URL). For example, copy a file from Amazon S3 Cloud to the folder `myFolder`.

```
mkdir myFolder
copyfile s3://bucketname/path_to_file/my_image.jpg myFolder
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

### **dbup and dbdown Commands: Switch between workspaces with one step**

When you debug your code, you can now change the current workspace and function context to any workspace and function context on the stack with one step. Specify the number of levels you want to move on the stack as an input argument to the `dbup` and `dbdown` commands.

### **bin2dec and hex2dec Functions: Convert text that includes binary or hexadecimal prefixes and suffixes**

The `bin2dec` and `hex2dec` functions convert text inputs that include the same prefixes and suffixes used for writing binary and hexadecimal literals.

For example, these calls to `bin2dec` return the same value:

```
bin2dec('111')
bin2dec('0b111')
bin2dec('0b111s32')
```

Similarly, these calls to `hex2dec` return the same value:

```
hex2dec('FF')
hex2dec('0xFF')
hex2dec('0xFFs32')
```

For more information on the syntax for using these prefixes and suffixes, see [Hexadecimal and Binary Values](#).

## dec2bin and dec2hex Functions: Convert negative numbers

The `dec2bin` and `dec2hex` functions convert negative numbers using their two's complement binary values.

For example, these calls to `dec2bin` and `dec2hex` convert negative numbers.

```
dec2bin(-1)
ans =
    '11111111'
dec2bin(-16)
ans =
    '11110000'
dec2hex(-1)
ans =
    'FF'
dec2hex(-16)
ans =
    'F0'
```

## complex Function: Create sparse complex arrays

You can now create sparse complex arrays from sparse input arguments using the `complex` function.

## Enumeration classes: Hide member names for compatible name changes

The enumeration block `Hidden` attribute enables enumeration class authors to hide member names from class users. Hiding enumeration members enables you to replace existing names with new names without introducing code incompatibilities. For more information, see [Hide Enumeration Members](#).

## matlab.mixin.SetGet: Set priority for partial property name matching

Classes that derive from `matlab.mixin.SetGet` can use the `PartialMatchPriority` property attribute to specify a relative priority for partial name matching. MATLAB applies this attribute when resolving incomplete and case-insensitive text strings that match more than one property name. For more information, see [Set Priority for Matching Partial Property Names](#).

## Class logical conversions: Support logical conversion more flexibly when writing classes

When MATLAB requires a logical value for expressions like `if` and `while` statements, it attempts a direct conversion using the `logical` converter function. If the result of the conversion is a nonlogical

value, then MATLAB attempts to call the `cast` function with the "like" flag. If the class of the object being converted defines a `cast` method that supports the "like" flag, and this method returns a logical value, then MATLAB uses this logical value. Otherwise, MATLAB throws a `MATLAB:invalidConversion` error.

## Functionality being removed or changed

### File Operations: Wildcard expression \*.\* on UNIX platforms matches only files that have an extension

#### *Behavior change*

Starting in R2020a, on UNIX® platforms, the wildcard expression \*.\* no longer matches folders or files without an extension. In previous releases, the expression matches folders or files regardless of extension, including files without an extension. This change of behavior does not apply to Microsoft Windows platforms. This change affects the functions `copyfile`, `delete`, `dir`, `movefile`, and `rmdir`.

### copyfile Function: Symbolic links are treated consistently on platforms and file systems

#### *Behavior change*

Symbolic links (or symlinks) are file system objects that point to target files or folders. Starting in R2020a, the behavior of the `copyfile` function changes when operating on symlink files or folders.

- **Copy outcome is platform independent:** `copyfile` now treats symlinks on different operating systems in the same way. For example, consider a folder structure with a file `myFile.m` and a symlink pointing to `myFile.m`, specified as `symlinkToMyFile`.

```
myFile.m
symlinkToMyFile
```

Starting in R2020a, `copyfile('symlinkToMyFile','newFile')` copies the target of the symlink (that is, `myFile.m`) to the destination. In previous releases, on Linux, `copyfile` copies the symlink instead.

Platform	newFile (Starting in R2020a)	newFile (R2019b and Earlier)
Linux	myFile.m	symlinkToMyFile
Mac	myFile.m	myFile.m
Windows	myFile.m	myFile.m

- **copyfile copies only the contents of the source folder:** When copying a nonempty folder to a symlink folder, `copyfile` now copies the contents of the source folder (that is, the files and folders within the source folder) rather than the entire source folder. Similarly, when copying a symlink folder to a destination folder, only the contents of the symlink folder are copied.

For example, consider a folder structure on Linux consisting of a nonempty folder `myFolder` and a symlink to that folder named `symlinkToMyFolder`. This table shows the folder structure after the execution of `copyfile('myFolder','simlinkToMyFolder')` in different MATLAB releases.

Folder Structure Before Copy	Folder Structure After Copy (Starting in R2020a)	Folder Structure After Copy (R2019b and Earlier)
myFolder myFile.m symlinkToMyFolder	myFolder myFile.m symlinkToMyFolder myFile.m	myFolder myFile.m symlinkToMyFolder myFolder myFile.m

### Toolbox folders renamed: Old names will be removed

*Still runs*

Various toolbox folders have been renamed. The old toolbox folder names passed to the `toolboxdir`, `ver`, and `verLessThan` functions will be removed in a future release. Use the new names instead.

Toolbox	Old Folder Name	New Folder Name
Parallel Computing Toolbox	distcomp	parallel
Fixed-Point Designer™	fixpoint	fixedpoint
Simulink Real-Time™	xpc	slrt
Simscape™ Electrical™	powersys	sps

### System object authoring: StringSet will be removed

*Still runs*

The class `matlab.system.StringSet` will be removed in a future release to bring System object™ authoring closer to MATLAB classes.

System object syntax being removed	Migration
Enumerated properties with <code>matlab.system.StringSet</code>	Replace StringSets with enumerations or property validators. See Limit Property Values to Finite List.

### System object authoring: Logical and Positive Integer property attributes will be removed

*Still runs*

The System object property attributes `Logical` and `Positive Integer` will be removed in a future release to bring System object authoring closer to MATLAB classes.

System object syntax being removed	Migration
Property attributes <code>Logical</code> and <code>Positive Integer</code>	Replace these property attributes with property validators. See Validate Property and Input Values.

### System object authoring: Several `matlab.system.mixin.*` classes will be removed

*Still runs*

These mixin classes will be removed in a future release:

- `matlab.system.mixin.CustomIcon`
- `matlab.system.mixin.Nondirect`
- `matlab.system.mixin.Propagates`

- `matlab.system.mixin.SampleTime`

To simplify System object authoring, the functionality and methods from these classes are directly included with the base System object class `matlab.System`.

System object syntax being removed	Migration
<ul style="list-style-type: none"> <li>• <code>matlab.system.mixin.CustomIcon</code></li> <li>• <code>matlab.system.mixin.Nondirect</code></li> <li>• <code>matlab.system.mixin.Propagates</code></li> <li>• <code>matlab.system.mixin.SampleTime</code></li> </ul>	Remove any inheritance statements to these classes from the beginning of your System object class.

### **ismethod Function: String and character vector in first input argument will be treated as object**

*Behavior change in future release*

In a future release, the `ismethod` function will treat a string or character vector in the first input argument as a `string` or `char` object, instead of as the name of a class. To list the methods of a class by referring to the class by name, use the `methods` function. To determine if a specific method name is the name of a method of a class, use an expression like this:

```
any("methodName" == string(methods("ClassName")))
```

### **Constant properties: Defining a set or get access method for a constant property causes an error**

In previous releases, creating a set or get access method for a class property defined with the `Constant` attribute resulted in a warning. MATLAB ignored these methods. Starting in R2020a, MATLAB throws an error if a class defines access methods for `Constant` properties.

### **Constant properties: Specifying the Dependent attribute for a Constant property causes an error**

In previous releases, specifying the `Dependent` attribute for a class property defined with the `Constant` attribute resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties using both the `Constant` and `Dependent` attributes.

### **Property attributes: Specifying the Static attribute for a property causes an error**

In previous releases, specifying the `Static` attribute for a class property resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties using the `Static` attribute.

### **Property and method attributes: Specifying the Visible attribute for a property or method causes an error**

In previous releases, specifying the `Visible` attribute for a class property or method resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties or methods using the `Visible` attribute.

### **Defining classes and packages: Using `schema.m` will not be supported in a future release**

*Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

### empty static method: Inputs must be numeric or logical

#### *Behavior change*

In previous releases, the `empty` static method accepted inputs that are convertible to numeric or logical values. Starting in R2020a, MATLAB throws an error if the inputs are not numeric, logical, or values derived from numeric or logical classes.

For example, in releases prior to R2020a, passing the scalar char `'b'` to `empty` produces a result based on the Unicode numeric equivalent for the character `b` (which is the number 98).

```
a = double.empty(0, 'b')
a =
    0×98 empty double matrix
```

Starting in R2020a, the `empty` method does not accept inputs that are not numeric or logical.

```
a = double.empty(0, 'b')
Error using double.empty
Value must be numeric or logical.
```

### Calling superclass constructor: Stricter syntax is enforced

#### *Behavior change*

Subclass calls to superclass constructors cannot be part of other expressions, be contained in conditional statements, or be made after references to the object. For more information on these restrictions, see [Subclass Constructors](#).

In previous releases, MATLAB did not identify certain syntaxes as calls to superclass constructors, and therefore, did not throw errors caused by calling superclass constructors incorrectly. Starting in R2020a, MATLAB more strictly enforces the correct syntax.

The following class code illustrates some of the cases where previous releases do not identify calls to the superclass constructor because of the incorrect use of parenthesis or brackets.

```
classdef MyClass < A & B
    methods
        function obj = MyClass
            obj = [obj@A] % Not identified as superclass constructor call in previous releases
            obj@B
            (obj@B) % Not identified as a duplicate call to B constructor in previous releases
            if 1
                end (obj@B) % Not identified as a conditional call in previous releases
            end
            (obj@A) % Not identified as called after object referenced in previous releases
        end
    end
end
```

## Data Analysis

### Live Editor Tasks: Interactively manipulate tables and timetables, and generate code

Use Live Editor tasks to stack, unstack, or synchronize tables and timetables. Interactively explore the effects of your changes in output tables and timetables. The tasks also automatically generate code that becomes part of your live script.

In R2020a, MATLAB offers four tasks for manipulating data in tables and timetables:

- **Retime Timestep** — Resample or aggregate timestep data.
- **Stack Table Variables** — Combine values from multiple table variables into one table variable.
- **Synchronize Timetables** — Retime and combine timetables to new time vector.
- **Unstack Table Variables** — Distribute values from one table variable to multiple table variables.

To open tasks in the Live Editor, use the **Task** menu on the **Live Editor** tab. For more information, see Add Interactive Tasks to a Live Script.

### Basic Fitting Tool: Fit lines to plotted data using modernized interface

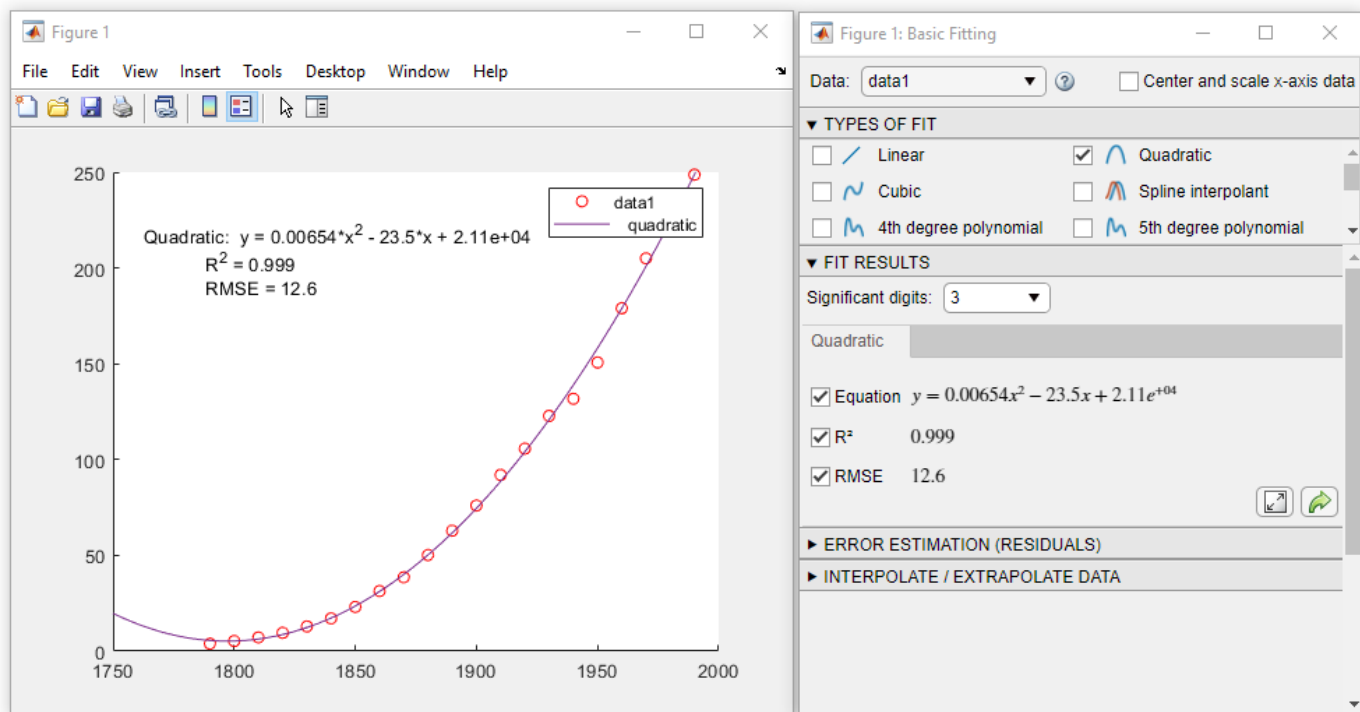
Fit lines to plotted data using the modernized interface of the **Basic Fitting** tool. Open the tool by selecting **Tools > Basic Fitting** from the figure toolbar.

For example, plot sample data.

```
load census
scatter(cdate, pop, 'r')
```

Open the **Basic Fitting** tool. Select a fit and display the equation,  $R^2$  value, and root mean square error (RMSE) value on the plot.





## detrend Function: Ignore NaN values

You can now ignore NaN values when computing trends with the `detrend` function using the `'omitnan'` parameter.

## accumarray Function: Maintain consistent output order on all platforms

The `accumarray` function now returns results in the same order as they appear in the input on all platforms.

Previously, if you used a function with `accumarray` that depended on the data order, such as `@(x) x(1)`, different platforms sometimes returned different results.

## leapseconds Function: List all leap seconds used by the datetime data type

To list all the leap seconds supported by the `datetime` data type, use the `leapseconds` function. The output table that lists the leap seconds includes the dates on which they occurred, their signs, and the cumulative adjustments. To determine the International Earth Rotation Service (IERS) Bulletin C version number of the leap second data being used in MATLAB, also use the second output argument. For more information, see IERS Bulletins.

To list leap seconds, use either of the following syntaxes:

```
T = leapseconds
[T,vers] = leapseconds
```

## timezones Function: Determine IANA Time Zone Database version

To determine the Internet Assigned Numbers Authority (IANA) Time Zone Database version supported by the `datetime` data type, use the second output argument of the `timezones` function.

```
[T,vers] = timezones
```

For more information on the IANA Time Zone Database, see [IANA Time Zone Database](#).

## renamevars Function: Rename variables in table or timetable

You can rename variables in a table or timetable using the `renamevars` function.

## rows2vars and unstack Function: Use naming rule to allow table and timetable variable names with any characters

Starting in R2020a, you can specify the rule for naming table and timetable variables when you use the `rows2vars` or `unstack` functions. Specify the rule using the 'VariableNamingRule' name-value pair argument.

Value of 'VariableNamingRule'	Rule
'modify' (default)	Modify variable names to be valid MATLAB identifiers
'preserve'	<p>Preserve original names that can have any Unicode characters, including spaces and non-ASCII characters.</p> <p><b>Note:</b> In some cases, the function must modify original names even when 'preserve' is the rule. Such cases include:</p> <ul style="list-style-type: none"> <li>• Duplicate names</li> <li>• Names that conflict with table dimension names</li> <li>• Names that conflict with a reserved name.</li> <li>• Names whose lengths exceed the value of <code>namelengthmax</code>.</li> </ul>

Previously, these functions modified table and timetable variable names when necessary so that such names were always valid MATLAB identifiers.

## containsrange, overlapsrange, and withinrange Functions: Determine if timetable row times intersect specified time range

To determine if the row times of a timetable intersect a specified time range, use one of these functions.

Function	Purpose
<code>containsrange</code>	Determine if timetable row times contain specified time range
<code>overlapsrange</code>	Determine if timetable row times overlap specified time range
<code>withinrange</code>	Determine if timetable row times are within specified time range

## tall Arrays: Operate on tall arrays with more functions, including `groupfilter` and `matches`

The functions listed here now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

- `groupfilter`
- `matches`
- `renamevars`

In addition, some functions have updated or removed limitations with tall arrays.

Function	Changes
<code>ismember</code>	<p><code>ismember</code> now allows both inputs to be tall arrays, as long as one of the inputs is the result of a reduction operation along the first dimension (such as <code>sum</code> or <code>mean</code>).</p> <p>Previously, only one input could be a tall array.</p>
<code>vartype</code>	<p>Many tall data analysis functions can now use <code>vartype</code> to select data and grouping variables:</p> <ul style="list-style-type: none"> <li>• Preprocessing functions with a 'DataVariables' name-value pair (such as <code>fillmissing</code>, <code>ischange</code>, and <code>smoothdata</code>) can use <code>vartype</code> to select the data variables.</li> <li>• Grouping functions with <code>groupvars</code> and <code>datavars</code> arguments (such as <code>grouptransform</code> and <code>groupsummary</code>) can use <code>vartype</code> to select the data and grouping variables.</li> </ul> <p>Previously, these functions could not use <code>vartype</code> to select data or grouping variables.</p>

Function	Changes
groupsummary and grouptransform	<p>groupsummary and grouptransform can now operate on multidimensional variables in a tall table or tall timetable.</p> <p>Previously, these functions could only operate on column vector variables.</p>

## Functionality Being Removed or Changed

### The datetime function 'InputFormat' month formats M and MM do not recognize names, and MMM does not recognize abbreviations

#### *Behavior change*

Starting in R2020a, when you specify month formats in the 'InputFormat' name-value pair argument of the `datetime` function, the M and MM formats do not recognize month names. Also, the MMM format recognizes only abbreviated names. In previous releases, all these formats recognized both abbreviated and full month names.

For more information on formats, see the `Format` property of the `datetime` function.

### Default aggregation function for nonnumeric data in unstack

#### *Behavior change*

In R2020a, if you do not specify the 'AggregationFunction' name-value pair argument of the `unstack` function, then the default aggregation function for nonnumeric data is the `unique` function. In previous releases, there was no default aggregation function for nonnumeric data, so `unstack` would raise an error.

### Behavior changes when the aggregation function has no data to aggregate in unstack

#### *Behavior change*

In R2020a, there are behavior changes when the aggregation function specified by the 'AggregationFunction' name-value pair argument of the `unstack` function has no data to aggregate. This situation can occur when there are no data values that correspond to values in the indicator variable after unstacking. In such cases, `unstack` essentially calls the aggregation function on an empty array.

For more information on this change in behavior, see the `Compatibility Considerations` section of `unstack`.

## Data Import and Export

### **Datstores: Write data from datastore to files using writeall**

You can write data from a datastore to files on disk using the `writeall` function.

Additionally, to add `writeall` functionality to custom datstores, you can use the new classes `matlab.io.datastore.FileWritable` and `matlab.io.datastore.FoldersPropertyProvider`. For more information, see [Add Support for Writing Data](#).

### **Datstores: Return timetables from tabularTextDatastore and spreadsheetDatastore objects**

`tabularTextDatastore` and `spreadsheetDatastore` objects have two new properties that enable you to work with timetables: `OutputType` and `RowTimes`. These properties specify whether the `read`, `readall`, and `preview` methods return tables or timetables.

Previously, these properties were only available in Parquet datstores.

### **Datstores: Partition and shuffle TransformedDatastore and CombinedDatastore objects**

You can now partition and shuffle arbitrarily nested transformations and combinations of datstores, subject to these conditions:

- You can partition and shuffle a `TransformedDatastore` object only if all of its underlying datstores can be partitioned and shuffled.
- You can partition and shuffle a `CombinedDatastore` object only if `subset` can be applied to all of its underlying datstores. The underlying datstores can also be transformations or combinations of datstores that can have `subset` applied to them.

Use `isPartitionable` and `isShuffleable` to test whether a `CombinedDatastore` object or a `TransformedDatastore` object can be partitioned or shuffled, and to determine when certain combinations of datstores are fit for parallel processing. `isPartitionable` and `isShuffleable` return `true` when the underlying datstores can be partitioned or shuffled, respectively.

### **Datstores: Process files and blocks within files iteratively using FileSet and BlockedFileSet objects**

You can process a large collection of files when moving through the files iteratively using the `matlab.io.datastore.FileSet` object. Similarly, you can process a large collection of blocks within files iteratively using the `matlab.io.datastore.BlockedFileSet` object.

### **Parquet Files: Control encoding scheme and Parquet version when writing files**

The `parquetwrite`, `parquetinfo`, and `write` functions have two new name-value pairs:

- 'VariableEncoding' controls whether a Parquet file uses plain or dictionary encoding for each variable.
- 'Version' specifies whether to use Parquet 1.0 or Parquet 2.0 file formatting.

## Text and Spreadsheet Files: Append, overwrite, or replace data using 'WriteMode' parameter

You can choose to append, overwrite, or replace data when writing to text and spreadsheet files by using the `WriteMode` parameter with these functions:

- `writetable`
- `writetimetable`
- `writematrix`
- `writecell`

## readtable Function: Uses results of detectImportOptions function by default

Starting in R2020a, the `readtable` function uses the results of the `detectImportOptions` function to import tabular data. In essence, these two `readtable` function calls behave identically.

```
T = readtable(filename)
T = readtable(filename,detectImportOptions(filename))
```

## Version History

There are several differences between the default behavior of `readtable` and its default behavior in previous releases. To call `readtable` with the default behavior it had up to R2019b, use the 'Format', 'auto' name-value pair argument.

```
T = readtable(filename,'Format','auto')
```

The table lists significant differences between the default behavior of `readtable` in R2020a and its default behavior in previous releases.

Description of Input Fields or Rows	Default R2020a readtable Behavior	Default Behavior in Previous Releases
First row does not have text to assign as names of output table variables	Assigns the names <code>Var1, . . . , VarN</code> as the names of output table variables	Converts the values in the first row of data values to the names of output table variables
Multiple rows of text as header lines	<ul style="list-style-type: none"> <li>• Ignore additional header lines</li> <li>• Import values in remaining rows as detected data types</li> </ul>	<ul style="list-style-type: none"> <li>• Import additional header lines as text in the first rows of output table</li> <li>• Import values in remaining rows as text</li> </ul>
Empty fields	Treat as missing values for detected data type	Treat as empty character vectors or strings
Values in quotes	Treat as detected data type	Treat as text

Description of Input Fields or Rows	Default R2020a readable Behavior	Default Behavior in Previous Releases
Text that cannot be converted	Treat as missing values for detected data type	Treat as text
Nonnumeric character trails numeric character without delimiter between them	Treat characters as nonnumeric	Treat numeric and nonnumeric characters as though delimiter separated them
Input text file has lines with different number of delimiters	Returns output table with extra variables	Raises error message

## textscan, readtable, detectImportOptions, and setvaropts Functions: Read and import hexadecimal and binary literals

Hexadecimal and binary literals are now supported in the functions `textscan`, `readtable`, `detectImportOptions`, and `setvaropts`. This list outlines the added capabilities of each function.

- `textscan` — Use the format specifier `'%x'` to read in data stored as hexadecimal data, and use `'%b'` to read in data stored as binary data. The default data type is `uint64`. For example, `res = textscan('110101', '%b')` will return `res = 53` with the default data type `uint64`.
- `readtable` — Text that is prefixed with the characters `'0x'` is now treated as hexadecimal data and text with the prefix `'0b'` is treated as binary data.
- `detectImportOptions` — Use the name-value pairs `'HexType'` or `'BinaryType'` to convert the data from hexadecimal or binary to decimal, then set the data type of the output data. For example, `'HexType', 'uint8'` converts prefixed hexadecimal data to decimal, then sets the data type of the output to an 8-bit unsigned integer during import.
- `setvaropts` — Set the data type and number system to be used when importing variables by using these name-value pairs:
  - `'Type'` sets the data type of the resulting output variable. For example, `'Type', 'uint32'` sets the data type of the output variable to a 32-bit unsigned integer.
  - `'NumberSystem'` converts the number system of the input variable from hexadecimal or binary to the decimal number system. For example, `'NumberSystem', 'hex'` converts data that is stored as hexadecimal to decimal data. If the number system of the input variable is specified as `'decimal'`, then no conversion is applied. `NumberSystem` is also a new property of the `NumericVariableImportOptions` object.

## h5read and h5readatt: Read non-scalar string data as MATLAB string arrays

The high-level HDF5 functions `h5read` and `h5readatt` now return HDF5 string arrays as MATLAB string arrays rather than cell arrays of character vectors. Single (scalar) HDF5 strings are still returned as MATLAB character vectors.

## h5create and h5write: Write string data to HDF5 files

You can now write string data to HDF5 files using `h5create` and `h5write` instead of using low-level HDF5 functions. String data can be specified as MATLAB character vectors or MATLAB string arrays.

## CDF Library: Upgraded to v3.7.0

The CDF library has been upgraded to version 3.7.0.

## Tiff Object: Read and write the values of the Rational Polynomial Coefficients tag

You can now read and write the values of the Rational Polynomial Coefficient (RPC) tag using the `RPCCoefficientTag` tag for the Tiff object. For more information, see Table 6 in *Exporting to Images*.

## jsonencode: Customize encoding in MATLAB classes

You can overload the `jsonencode` function to customize the JSON encoding for a user-defined MATLAB class. For an example, see *Customize JSON Encoding for MATLAB Classes*.

## jsonencode: Encode enumerations

`jsonencode` encodes enumerations as strings. For example,

```
on = matlab.lang.OnOffSwitchState.on;  
jsonencode(on)
```

```
ans =
```

```
    'on'
```

## Functionality being removed or changed

### readtable, writetable, textscan, and similar functions use automatic character set detection and UTF-8 encoding by default

*Behavior change*

As of R2020a, most functions that read text data use automatic character set detection to detect the character encoding. Functions that use automatic character set detection include `fileread`, `textscan`, `readvars`, `readtable`, `readcell`, `readmatrix`, and `readtimetable`.

Similarly, most functions that write text data use UTF-8 as the default character encoding. Using UTF-8 provides interoperability between all platforms and locales without data loss or corruption. Functions that use UTF-8 encoding by default include `writematrix`, `writetable`, `writecell`, and `writetimetable`.

### File I/O functions, such as fscanf and fprintf, use automatic character set detection and UTF-8 encoding by default

*Behavior change*

As of R2020a, character-oriented file I/O functions such as `fscanf`, `fgets`, and `fgetl` trigger automatic character set detection when reading a file that was opened using `fopen` without a specified encoding.

Similarly, `fprintf` defaults to using UTF-8 encoding when writing a file that was opened using `fopen` without a specified encoding.



**h5write and h5writeatt use UTF-8 character encoding by default***Behavior change*

UTF-8 is now the default character encoding for the high-level HDF5 functions `h5write` and `h5writeatt` to ensure that all Unicode code points can be correctly represented in HDF5 files.

**h5read and h5readatt return non-scalar string data as MATLAB strings***Behavior change*

The high-level HDF5 functions `h5read` and `h5readatt` now return HDF5 string arrays as MATLAB string arrays rather than cell arrays of character vectors. Single (scalar) HDF5 strings are still returned as MATLAB character vectors.

**web function does not return a handle or URL for pages that open your system browser***Behavior change*

The `web` function does not return a handle or URL for pages that open in the system browser. This includes all external pages, which by default open in your system browser, unless configured otherwise in the MATLAB Web Preferences.

To update your code, remove the handle and URL output arguments from instances of the `web` function. This table shows examples of how you can update your code.

Before	After
<code>[stat,h] = web('https://www.mathworks.com', stat, 'browser')</code>	<code>stat = web('https://www.mathworks.com', '-browser')</code>
<code>[stat,h,url] = web('https://www.mathworks.com', stat, 'browser')</code>	<code>stat = web('https://www.mathworks.com', '-browser')</code>

**hdftool has been removed***Errors*

`hdftool` has been removed. To programmatically import HDF4 or HDF-EOS files, use the `hdfread` function instead.

**Importing HDF5 files using the Import Tool is no longer supported***Behavior change*

The Import Tool no longer supports importing HDF5 files. To programmatically import HDF4 or HDF-EOS files, use the `hdfread` function instead.

## Mathematics

### **nufft and nufftn Functions: Compute nonuniform fast Fourier transforms**

To compute 1-D or N-D fast Fourier transforms with nonuniform sampling, use the `nufft` and `nufftn` functions, respectively.

### **sparse Function: Support for integer subscripts and logical aggregation**

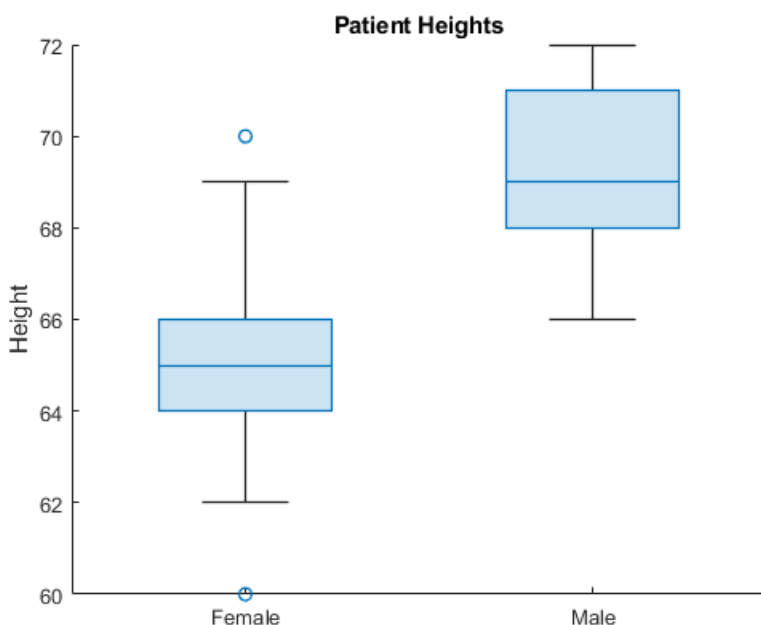
The `sparse` function has two new capabilities:

- When you construct a sparse matrix using the syntax `sparse(i, j, v)`, the subscript inputs `i` and `j` can now be integer data types.
- When the third input of the syntax `sparse(i, j, v)` contains logical values and there are repeated subscripts in `i` and `j`, the `sparse` function now applies a logical any operation to the values with repeated subscripts.

## Graphics

### boxchart Function: Visualize grouped numeric data by using box charts

To create box charts, also called box plots, use the `boxchart` function. For each group of data, the corresponding box chart displays the following information: the median, first and third quartiles, outliers (computed using the interquartile range), and nonoutlier minimum and maximum values. When you use vector data, you can use the first input argument `xgroupdata` to split your data into groups and specify the positions of the corresponding boxes. When you use matrix data, `boxchart` creates a separate box for each column in the matrix.



### exportgraphics and copygraphics Functions: Save and copy graphics with improved support for publishing workflows

Use the `exportgraphics` function to save the contents of any axes, figure, chart that can be a child of a figure, tiled chart layout, or container such as a panel. This function provides a better alternative to the `print` and `saveas` functions when you want to:

- Save graphics displayed in an app or in MATLAB Online.
- Minimize the white space around the content.
- Save a PDF fragment with embeddable fonts.
- Save a subset of the content in a figure.
- Control the background color.

The `copygraphics` function provides much of the same functionality as the `exportgraphics` function, except that it copies the content to your system clipboard instead of saving it to a file. Use this function to copy and paste content from MATLAB into other applications.

## ChartContainer Class: Develop charts that display a tiling of Cartesian, polar, or geographic plots

Charts you develop as a subclass of `matlab.graphics.chartcontainer.ChartContainer` now provide a `TiledChartLayout` object, which you can use to arrange one or more Cartesian axes, polar axes, or geographic axes in your chart.

To access the `TiledChartLayout` object, call the `getLayout` method. To place one or more axes objects into the layout, call the `axes`, `polaraxes`, or `geoaxes` function, and specify the `TiledChartLayout` object as the first input argument. For more information, see [Develop Charts With Polar Axes, Geographic Axes, or Multiple Axes](#).

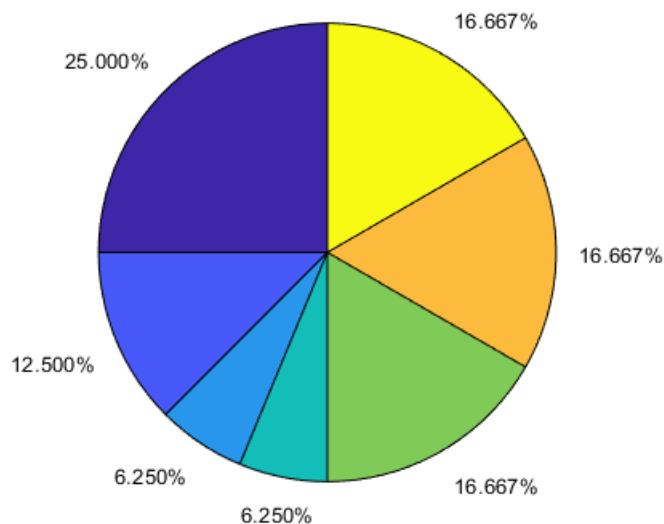
## Tiled Chart Layout: Position, nest, and change the grid size of layouts

Customize layouts you create with the `tiledlayout` function by setting these properties on the `TiledChartLayout` object:

- `Position`, `InnerPosition`, `OuterPosition`, and `PositionConstraint` — For changing the size and location of the layout.
- `GridSize` — For changing the number of tiles along the rows and columns of the layout. You can set this property only when the layout is empty.
- `Layout` — For configuring nested layouts. Nested layouts consist of at least two `TiledChartLayout` objects, where the `Parent` property of one object is the other object.

## pie Function: Specify a numeric format for the percentage labels

Specify the numeric format for the percentage labels on a pie chart. For example, you can specify the number of decimal places or significant digits to display.



## **Axes Convenience Functions: Pass an array of axes or chart objects to convenience functions such as grid, hold, and box**

Modify multiple axes or charts at one time by passing a vector of objects to the `grid`, `hold`, `box`, `xlabel`, `ylabel`, `zlabel`, and `title` functions.

## **SeriesIndex and NextSeriesIndex Properties: Control how plots cycle through colors and line styles**

Set the `SeriesIndex` property on plot objects such as `Line`, `Scatter`, and `Bar` to control how the objects vary in color and possibly line style. By default, the `SeriesIndex` for an object is a number that corresponds to the object's order of creation. MATLAB uses the number to calculate indices into the `ColorOrder` and `LineStyleOrder` properties of the axes. Changing the value of this property is useful when you want to reassign the colors or line styles of the objects in the axes.

The `NextSeriesIndex` property on the axes maintains a count of the objects that have a `SeriesIndex` property. MATLAB uses it to assign the value of the `SeriesIndex` property for each new object in the axes. The count starts at 1 and increments for each additional object. The `NextSeriesIndex` property is useful when you want to track how the objects cycle through the colors and line styles.

For more information about controlling colors and line styles, see [Control Colors, Line Styles, and Markers in Plots](#).

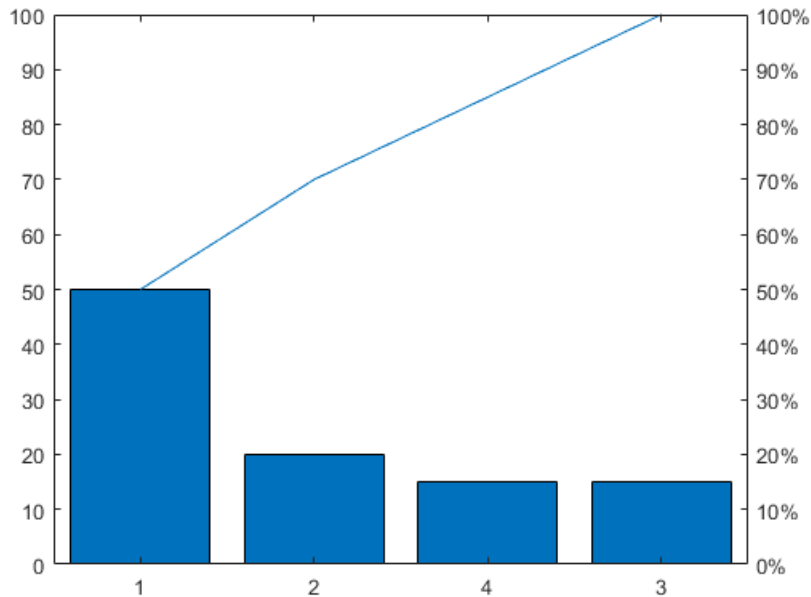
## **colororder Function: Control colors in scatter histograms and parallel plots**

The `colororder` function now supports charts created with the `scatterhistogram` and `parallelplot` functions.

## **pareto Function: Specify the fraction of the cumulative histogram to include**

Specify the fraction of the cumulative histogram to display in a Pareto chart as the last argument to the `pareto` function. For example, this Pareto chart includes all the bars that make up 100% of the cumulative histogram of `Y`.

```
Y = [50 20 15 15];  
pareto(Y,1)
```





## Axes: Control margins for titles and labels by setting the `InnerPosition` and `PositionConstraint` properties

Set the `InnerPosition` property on any type of axes object to control the size and location of the plot box.

Set the `PositionConstraint` property of an axes object or a chart that can be a child of a figure to control the space around the plot box when you add or modify decorations such as titles and axis labels. This property is similar to the existing `ActivePositionProperty`. However, unlike `ActivePositionProperty`, the `PositionConstraint` property accepts the values 'outerposition' and 'innerposition' instead of 'outerpositon' and 'position'.

## Built-In Axes Interactions: Explore data with cursors that show available interactions

When you hover within a set of axes, the cursor changes to indicate when you can rotate axes, create data tips, and pan axis rulers. For example, when you hover over ruler labels in 2-D axes, the cursor indicates that you can drag to pan the axis ruler by changing to . When you hover over plots that support data tips, the cursor indicates that you can click to create a data tip by changing to . You can always drag to pan within 2-D axes and scroll to zoom, even when the cursor indicates another interaction.

You can disable these cursor changes by setting the `Pointer` property of the figure.

```
scatter(1:10,1:10)
f = gcf;
f.Pointer = 'arrow';
```


## Built-In Axes Interactions: Customize built-in interactions on geographic axes

By default, geographic axes include built-in pan, zoom, and data tip interactions. Enable and disable the default built-in interactions for `GeographicAxes` objects using the `enableDefaultInteractivity` and `disableDefaultInteractivity` functions.

You can create a customized set of built-in interactions by setting the `Interactions` property of a `GeographicAxes` object. Specify the `Interactions` property as an array of `ZoomInteraction`, `PanInteraction`, or `DataTipInteraction` objects. For more information, see [Control Chart Interactivity](#).

## linkdata Function: Open dialog box to specify data sources using new syntax

The `linkdata` function has a new syntax, `linkdata showdialog`, that opens the **Linked Plot Data Sources** dialog box. Use the dialog box to interactively specify data sources for plotted data.

Alternatively, instead of calling `linkdata showdialog`, you can open the dialog box by clicking **Link/Unlink Plot**  in the figure toolbar.

## Functionality being removed or changed

### Most properties that accept the values 'on' or 'off' now return an OnOffSwitchState value *Behavior change*

Most graphics object properties that accept the values 'on' or 'off' now accept and return a `matlab.lang.OnOffSwitchState` value. For example, you can specify the `Visible` property of an axes object as 'on', 'off', 1, 0, or a logical value. A value of 'on' is equivalent to `true`, and 'off' is equivalent to `false`. Thus, you can use the value of the property as a logical value in a conditional statement.

Any code that sets a property to 'on' or 'off' behaves the same way as in previous releases. However, you might need to update code that checks the value of the property. This table describes the most common situations.

Coding Pattern	Example of Coding Pattern	Updated Code
Checking for a specific character	<code>obj.Visible(2) == 'n'</code>	<code>obj.Visible == 1</code>
Checking the data type	<code>isequal(class(obj.Visible), 'char')</code> or <code>ischar(obj.Visible)</code>	<code>isequal(class(obj.Visible), ... 'matlab.lang.OnOffSwitchState')</code> or <code>isa(obj.Visible, ... 'matlab.lang.OnOffSwitchState')</code>
Checking the value in a unit test	<code>verifyEqual(testcase,obj.Visible)</code>	<code>verifyEqual(obj.Visible, ... matlab.lang.OnOffSwitchState.on)</code>

### ActivePositionProperty is not recommended

*Still runs*

Setting or getting `ActivePositionProperty` on an axes or chart object is not recommended. Use the `PositionConstraint` property instead.

There are no plans to remove `ActivePositionProperty` at this time, but the property is no longer listed when you call the `set`, `get`, or `properties` functions on an axes or chart object.

To update your code, make these changes:

- Replace all instances of `ActivePositionProperty` with `PositionConstraint`.
- Replace all references to the `'position'` option with the `'innerposition'` option.

For example, the following code sets the axes `ActivePositionProperty` to `'position'`.

```
ax = gca;  
ax.ActivePositionProperty = 'position';
```

Here is the updated code, which has the same effect.

```
ax = gca;  
ax.PositionConstraint = 'innerposition';
```

### **ChartContainer subclasses assign property values after the setup method runs**

#### *Behavior change*

When you create an instance of a `ChartContainer` subclass, and pass property name-value pair arguments to the constructor, the property values are assigned after the `setup` method runs. In R2019b, the property values are assigned before the `setup` method runs.

If the `setup` method of your class references the value of a property on the object, you can update your code in either of the following ways:

- Assign a default value for the property when you define it.
- Move the code that references the property to the `update` method.

### **Calling the ChartContainer.getAxes method returns an axes object as a child of a TiledChartLayout object**

#### *Behavior change*

When you call the `getAxes` method in a `ChartContainer` subclass, the method now returns an axes object that is a child of a `TiledChartLayout` object. If there are no axes in the chart, `getAxes` creates a Cartesian axes object. The chart no longer has an axes object until you create one by calling the `getAxes` method or one of the axes creation functions: `axes`, `polaraxes`, or `geoaxes`.

As a consequence of these changes, the axes in your chart might not be the current axes. Your code might produce unexpected results if you call the following types of functions within your class methods without specifying the target axes object.

- Plotting functions — For example, `plot`, `scatter`, `bar`, or `surf`
- Functions that modify the axes — For example, `hold`, `grid`, or `title`

In R2019b, the axes object is a child of the chart object, and it is the current axes within the scope of your class methods.

To update your code, specify the axes object as the first input argument when calling plotting functions and functions that modify the axes.



**Implementing callbacks on geographic plots disables built-in interactions***Behavior change*

Starting in R2020a, when you implement callbacks such as `ButtonDownFcn` on a geographic plot, MATLAB automatically disables built-in interactions.

In previous releases, implementing callbacks did not disable built-in interactions on geographic plots.

**Align Distribute Tool will be removed in a future release***Still runs*

The Align Distribute Tool will be removed in a future release.

To control the arrangement of multiple plots in a figure, create a tiled chart layout using the `tiledlayout` function instead.

To align or distribute graphics objects within a figure, select **Tools > Align** or **Tools > Distribute** from the figure toolbar instead.

**Charting functions return output only when you specify an output argument***Behavior change*

The `heatmap`, `geobubble`, `parallelplot`, `scatterhistogram`, `stackedplot`, `wordcloud`, `xline`, and `yline` functions no longer return the chart object as the `ans` variable when you call them without specifying an output argument. This new behavior is consistent with the behavior of most other charting functions.

In previous releases, the functions return the chart object as `ans` by default. If you have code that references a chart object that is stored in the `ans` variable, update your code by assigning the output to a different variable before referencing it.

## App Building

### **uicontextmenu Function: Add and configure context menu components in apps and on the App Designer canvas**

You can now create context menus in App Designer apps or in apps created with the `uifigure` function. When you right-click a UI component that has a context menu assigned to it, a list of menu items appears.

In apps created programmatically with the `uifigure` function, create a context menu using the `uicontextmenu` function. Add menu items to it using the `uimenu` function. Then, assign it to a component by setting the `ContextMenu` property of the component to the `ContextMenu` object.

In App Designer, create a context menu and assign it to a component by dragging it from the **Component Library** onto the component. For more details, see [Create and Edit Context Menus](#).

### **uitoolbar Function: Add custom toolbars to apps programmatically**

To programmatically add custom toolbars to your App Designer app or your app created with the `uifigure` function, use the `uitoolbar` function. Add push tools or toggle tools to the toolbar using the `uipushtool` or `uitoggletool` functions.

### **Icon Property: Display SVG, animated GIF, or truecolor image array icons in buttons and tree nodes**

The `Icon` property of `Button`, `ToggleButton`, and `TreeNode` objects now supports SVG and animated GIF files and truecolor image array data.

### **Mouse Pointer: Change the mouse pointer symbol in apps**

You can now change the mouse pointer symbol in apps created with the `uifigure` function or in App Designer to options such as `'hand'` or `'crosshair'`, or you can create your own pointer symbol.

For apps created with the `uifigure` function, set the `Pointer` property of the `Figure` object. For apps created with App Designer, select the component in the **Component Browser**. Then, in the **Inspector** tab, select a pointer from the **Pointer** drop-down menu.

To create a custom pointer symbol, programmatically set the `Pointer` property value to `'custom'`, and use the `PointerShapeCData` property to define the symbol. To specify the active pixel of a custom pointer symbol, set the `PointerShapeHotSpot` property.

For more information, see [UI Figure Properties](#).

### **Graphics Support: Create annotations, brush data, configure data tips, save and copy graphics**

You can now create annotations, brush data, configure data tips, and save or copy graphics in App Designer apps or in apps created with the `uifigure` function.

To create annotations, use the `annotation` function.

Use brush mode to mark chart data interactively. Then, you can remove or replace marked data values, or export values to the workspace. To turn on brush mode, select the data brushing button



from the axes toolbar or use the `brush` function to set the brush mode of the figure to 'on'.

Configure data tips in any of these ways:

- Edit data tip labels by double-clicking them.
- Display multiple pinned (persistent) data tips by clicking more than one data point in the plot.
- Click and drag pinned data tips to move their location with respect to the data point.
- View data tip options by right-clicking on a pinned data tip to display the context menu.

Save graphics displayed in an app using the `exportgraphics` or `copygraphics` function.

## GUIDE to App Designer Migration Tool for MATLAB: Migrate GUIDE apps to App Designer in less time and with fewer manual code updates

Improvements to the migration tool significantly reduce the time and number of manual code updates required to get your app running in App Designer. These improvements allow most of your app code to work in App Designer with few or no manual changes required. For more information, see [GUIDE Migration Strategies](#).

The GUIDE to App Designer Migration Tool for MATLAB is available through the Add-On Explorer in the MATLAB desktop or through File Exchange on MATLAB Central™.

## App Testing Framework: Perform press gestures with different selection types

The app testing framework supports mouse selection types in `press` gestures that are performed on figures created with the `uifigure` function. To specify the selection type, use the 'SelectionType' name-value pair argument. For example:

```
fig = uifigure;
testCase = matlab.uittest.TestCase.forInteractiveUse;
testCase.press(fig, 'SelectionType', 'open')
```

## Functionality Being Removed or Changed

### JavaContainer property will be removed in a future release

*Warns*

The `JavaContainer` property is undocumented and will be removed in a future release. Update your code to use documented alternatives. For a list of documented functionality that you can use instead, see [Recommendations for MATLAB Apps Using Java & ActiveX on mathworks.com](#).

### UIContextMenu property of graphics objects and UI components is not recommended

*Still runs*

Starting in R2020a, using the `UIContextMenu` property to assign a `ContextMenu` object to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The new property can have the same values as the old one.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

### Callback property of ContextMenu objects is not recommended

*Still runs*

Starting in R2020a, using the `Callback` property of a `ContextMenu` object is not recommended. To specify behavior when you interact with a context menu, use the `ContextMenuOpeningFcn` property of the `ContextMenu` object instead. The new property can reference callback functions in the same way as the old one.

There are no plans to remove support for the `Callback` property of `ContextMenu` objects at this time. However, this property no longer appears in the list returned by calling the `get` function on a `ContextMenu` object.

### Visible and Position properties of ContextMenu objects are not recommended

*Still runs*

Starting in R2020a, using the `Visible` and `Position` properties to configure a context menu to open at a specific location is not recommended. In apps created with the `uifigure` function, use the `open` function instead.

There are no plans to remove support for the `Visible` and `Position` properties of `ContextMenu` objects at this time. However, these properties no longer appear in the list returned by calling the `get` function on a `ContextMenu` object.

### Font size and color of column and row headers in table UI components has changed

*Behavior change*

Starting in R2020a, table UI components created in App Designer or in figures created with the `uifigure` function display column and row headers in a larger font size and darker font color. For example, this code that creates a table UI component with table array data renders differently in R2020a than it does in R2019b:

```
fig = uifigure;

dates = datetime([2016,01,17; 2017,01,20], 'Format', 'MM/dd/yyyy');
m = [10; 9];
tdata = table(dates,m, 'VariableNames', {'Date', 'Measurement'});

uit = uitable(fig, 'Data', tdata);
uit.RowName = 'numbered';
```

**R2020a:**

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

**R2019b:**

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

## Performance

### Live Editor Loop Execution: Improved performance when running loops in live scripts

Loops run significantly faster in live scripts. For example, a live script containing this code, which runs a loop one million times and displays the current loop iteration every ten thousand iterations, is approximately 42x faster.

```
for t = 1:1000000
    if ~mod(t, 10000)
        disp(t)
    end
end
```

The approximate execution times are:

**R2019b:** 2.291 s

**R2020a:** 0.054 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

The more iterations a loop contains, the greater the performance improvement becomes.

### Live Editor Animation Output: Improved performance when animating plots in live scripts

For-loop animations display faster in live scripts. For example, a live script containing this code, which creates a for-loop animation of a sine wave plot, is approximately 1.3x faster.

```
tic
x = 0:0.1:10*pi;
y = sin(x);
xlim([0 10*pi])
ylim([-1 1])
hold on
p = plot(x(1),y(1));
for k=1:length(x)
    p.XData = x(1:k);
    p.YData = y(1:k);
    drawnow
end
toc
```

The approximate execution times are:

**R2019b:** 8.875 s

**R2020a:** 6.633 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

## datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting

datetime, duration, and calendarDuration subscripted assignment is significantly faster. Performance is now essentially constant with the number of elements in an array.

- For example, when you assign into a datetime array with  $10^6$  elements, performance in R2020a is approximately 25x faster than in R2019b, as shown below.

```
function timingTest()
    dt = datetime + hours(1:1e6);
    indices = randi(1e6,1,10000);
    rhs = NaT;

    tic;
    for i = indices
        dt(i) = rhs;
    end
    toc
end
```

The approximate execution times are:

**R2019b:** 0.42 s

**R2020a:** 0.017 s

- Similarly, assignment into a duration array is faster. For example, when you assign into a duration array with  $10^6$  elements, performance in R2020a is approximately 11x faster than in R2019b.

```
function timingTest()
    d = hours(1:1e6);
    indices = randi(1e6,1,10000);

    tic;
    for i = indices
        d(i) = NaN;
    end
    toc
end
```

The approximate execution times are:

**R2019b:** 0.43 s

**R2020a:** 0.039s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function timingTest.

These performance improvements occur only when you make subscripted assignments within a function. There is no improvement when subscripting into datetime, duration, and calendarDuration arrays at the command line, or within try-catch blocks.

## datetime Data Type Format Parsing: Improved performance when parsing format of text inputs

`datetime` parsing performance is significantly faster when parsing the format of text inputs. For example, this code parses the date format of a string using the `datetime` function. The code executes approximately 1.75x faster in R2020a than in R2019b.

```
function timingTest()
    d1 = datetime(2010,1,1:10000);
    s = string(d1, 'dd-MMM-uuuu');
    tic
    for i = 1:100, d2 = datetime(s); end
    toc
end
```

The approximate execution times are:

**R2019b:** 5.44 s

**R2020a:** 3.10 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

## table Data Type Indexing: Improved performance when assigning elements by subscripting into table variables

`table` subscripted assignment into table variables is significantly faster. Performance is essentially constant with the number of elements in each table variable.

- For example, when you use dot indexing to assign elements to a variable with  $10^6$  elements, performance in R2020a is approximately 2x faster than in R2019b, as shown below.

```
function timingTest()
    t = table(zeros(1e6,1));
    indices = randi(1e6,1,10000);

    tic;
    for i = indices
        t.Var1(i) = rand;
    end
    toc
end
```

The approximate execution times are:

**R2019b:** 1.20 s

**R2020a:** 0.59 s

- Similarly, assignment using curly braces is faster. When you assign into three table variables with  $10^6$  elements, performance in R2020a is approximately 1.2x faster than in R2019b, as shown below.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
```

```
indices = randi(1e6,1,10000);  
  
tic;  
for i = indices  
    t{i,:} = rand;  
end  
toc  
end
```

The approximate execution times are:

**R2019b:** 8.04 s

**R2020a:** 6.68 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

The performance improvement occurs only when you make table subscripted assignments within a function. There is no improvement when subscripting into tables at the command line, or within `try-catch` blocks.

## Subscripted Reference: Improved performance for struct arrays stored in a property of an object

The performance of indexing into `struct` arrays that are stored in a property of a MATLAB object has improved. For example, this code executes about 2.3x faster in R2020a:

```
classdef ContainerClass  
    properties  
        field  
    end  
end  
  
function out = timingTest  
  
    M = struct('f', 1);  
    M(2) = struct('f', 2);  
    C = ContainerClass;  
    C.field = M;  
  
    tic  
    for j = 1:4e5  
        out = C.field(1);  
    end  
    toc  
  
end
```

The approximate execution times are:

**R2019b:** 1.14 s

**R2020a:** 0.5 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.



## imread Function: Improved performance in reading JPEG images

The `imread` function shows improved performance when reading JPEG images. The higher the image resolution (measured in pixels), the greater the performance improvement becomes.

For example, a JPEG image with a resolution of 5120x3840 is read about 2.1x faster in R2020a:

```
function out = testperformance(filename)
for ii = 1:100
    tic
    imread(filename);
    ts(ii) = toc;
end
out = mean(ts)
end
```

The approximate execution times are:

**R2019b:** 0.46s

**R2020a:** 0.22s

The code was timed on a Windows 10, Intel Xeon® CPU E5-1650 @ 3.6 GHz test system by calling the function `testperformance`.

## readmatrix Function: Improved performance in reading data

The `readmatrix` function shows improved performance when reading matrix data. The larger the matrix to be read, the greater the performance improvement becomes.

For example, a matrix that is 30,000 x 30,000 elements in size is read about 1.1x faster in R2020a:

```
function out = readmatrix_performance()
M = randi(10,30000);
writematrix(M);
for ii = 1:10
    tic
    readmatrix('M.txt');
    ts(ii) = toc;
end
out = mean(ts);
```

The approximate execution times are:

**R2019b:** 225.21s

**R2020a:** 198.42s

The code was timed on a Windows 10, Intel Xeon® CPU E5-1650 @ 3.6 GHz test system by calling the function `readmatrix_testperformance`.

## ode15s, ode23t, and ode15i Solvers: Improved performance solving differential equations

The ode15s, ode23t, and ode15i solvers show improved performance solving differential equations. The performance improvement gets better as the number of linear systems evaluated by the solver during the solution process increases.

- For ode15s, this code executes about 3.5x faster in R2020a:

```
function timing0de15s
[t,y] = ode15s(@vdp1000,[0 1e5],[2; 0]);
end
```

The approximate execution times are:

**R2019b:** 2.83 s

**R2020a:** 0.82 s

- For ode23t, this code executes about 4.1x faster in R2020a:

```
function timing0de23t
[t,y] = ode23t(@vdp1000,[0 1e5],[2; 0]);
end
```

The approximate execution times are:

**R2019b:** 2.92 s

**R2020a:** 0.72 s

- For ode15i, this code executes about 2.3x faster in R2020a:

```
function timing0de15i
[y0,yp0] = decic(@weissinger,1,sqrt(3/2),1,0,0);
for k = 1:100
    [t,y] = ode15i(@weissinger,[1 10],y0,yp0);
end
end
```

The approximate execution times are:

**R2019b:** 0.51 s

**R2020a:** 0.22 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timing0de15s)
timeit(@timing0de23t)
timeit(@timing0de15i)
```

## transpose and ctranspose Functions: Improved performance on large arrays

The performance of the `transpose` and `ctranspose` functions (`.'` and `'` operators) has improved when operating on large arrays. For example, this code executes about 4.4x faster in R2020a when transposing a 10,000-by-10,000 matrix:

```
function timingTest
    rng default
    A = rand(1e4);
    tic
    At = A';
    toc
end
```

The approximate execution times are:

**R2019b:** 0.61 s

**R2020a:** 0.14 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.50 GHz test system by calling the function `timingTest`.

## ordschur and ordqz Functions: Improved performance operating on large matrices

The `ordschur` and `ordqz` functions show improved performance when operating on square matrices of order 200 or greater. The performance improvement gets better as the matrix gets larger.

- For `ordschur`, this code executes about 11.3x faster in R2020a:

```
function timingOrdschur
    rng default
    A = randn(2000);
    [U,S] = schur(A,'real');
    tic
    [U,S] = ordschur(U,S,'lhp');
    toc
end
```

The approximate execution times are:

**R2019b:** 3.5 s

**R2020a:** 0.31 s

- For `ordqz`, this code executes about 8.3x faster in R2020a:

```
function timingOrdqz
    rng default
    A = randn(2000);
    B = randn(2000);
    [A,B,Q,Z] = qz(A,B,'complex');
    tic
    [A,B,Q,Z] = ordqz(A,B,Q,Z,'lhp');
```

```
toc  
end
```

The approximate execution times are:

**R2019b:** 18.2 s

**R2020a:** 2.2 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the functions `timingOrdschur` and `timingOrdqz`.

## **sparse Function: Improved performance constructing sparse matrices**

The `sparse` function shows improved performance constructing sparse matrices. The performance improvement applies to all syntaxes of the function and gets better as the constructed matrix gets larger.

For example, this code executes about 4.4x faster in R2020a:

```
function timingSparse  
m = 1e5;  
n = 1e3;  
nz = 1e7;  
rng default  
i = randi(m,nz,1);  
j = randi(n,nz,1);  
v = rand(nz,1);  
  
tic  
A = sparse(i,j,v,m,n);  
toc  
end
```

The approximate execution times are:

**R2019b:** 1.68 s

**R2020a:** 0.38 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSparse`.

## **interp1 Function: Faster interpolation for small problem sizes**

The `interp1` function shows improved performance for problems with less than about 10,000 sample points. The improvement applies to all interpolation methods.

For example, this code executes about 2.5x faster in R2020a:

```
function timingInterp1  
x = 1:100;  
v = sin(x/3);  
xq = 1:0.5:100;  
  
for k = 1:10000
```

```

    vq = interp1(x,v,xq);
end
end

```

The approximate execution times are:

**R2019b:** 0.42 s

**R2020a:** 0.17 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingInterp1)
```

## assert Function: Improved performance for most common use cases

The performance of the `assert` function has significantly improved, making it virtually penalty-free to use assertions in error handling applications. For example, this code executes approximately 300x faster in R2020a:

```

function testAssertPerformance
x = -1;
for i = 1:1e6
    assert(x == -1,'Sample error message.')
end
end

```

The approximate execution times are:

**R2019b:** 0.30 s

**R2020a:** 0.001 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system using the `timeit` function:

```
timeit(@testAssertPerformance)
```

There is no performance improvement if `assert` is called with an error message that contains formatting conversion characters, such as those used with the MATLAB `sprintf` function.

## nexttile Function: Improved performance when creating several axes in a tiled chart layout

The `nexttile` function shows improved performance when creating several axes in a tiled chart layout. The performance improvement gets better as the number of axes increases.

For example, this code creates 100 axes in a 10-by-10 layout. It executes about 7.2x faster in R2020a.

```

function timingTest
tiledlayout(10,10);
for i = 1:100
    nexttile;
end
end

```

The approximate execution times are:

**R2019b:** 3.6 s

**R2020a:** 0.5 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingTest)
```

## App Designer Code View: Improved performance when displaying and editing code in App Designer

Displaying and editing code in the App Designer **Code View** editor is faster in R2020a than in R2019b. This affects actions like entering text, creating a new line of code, and adding and deleting functions.

In apps with approximately 2500 lines of code and 4500 lines of code, these types of improvements can be seen:

- Switching from **Design View** to **Code View** is about 1.5x faster in apps with approximately 2500 lines of code, and about 1.6x faster in apps with approximately 4500 lines of code.

The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
<b>R2019b</b>	0.63 s	1.10 s
<b>R2020a</b>	0.42 s	0.70 s

- Entering text in the **Code View** editor is about 2.8x faster in apps with approximately 2500 lines of code, and about 3.9x faster in apps with approximately 4500 lines of code.

The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
<b>R2019b</b>	3.97 s	6.67 s
<b>R2020a</b>	1.44 s	1.72 s

- Creating a new line of code by pressing **Enter** in the **Code View** editor is about 5.8x faster in apps with approximately 2500 lines of code, and about 9.4x faster in apps with approximately 4500 lines of code.

The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
<b>R2019b</b>	1.33 s	3.38 s
<b>R2020a</b>	0.23 s	0.36 s

- Adding and deleting a function in apps with approximately 2500 lines of code are about 1.7x and 2.9x faster, respectively.

The approximate execution times are:

Release	Add	Delete
R2019b	0.12 s	0.086 s
R2020a	0.07 s	0.030 s

Similarly, in apps with approximately 4500 lines of code, adding and deleting a function are about 1.8x faster and 2.8x faster, respectively.

The approximate execution times are:

Release	Add	Delete
R2019b	0.18 s	0.13 s
R2020a	0.10 s	0.046 s

These actions were timed on a *Windows 10, Intel Xeon® CPU E5-1650 v3 @ 3.5GHz* test system with *NVIDIA Quadro K620* graphics card.

## Graphics Rendering in UI Figures: Improved graphics rendering performance on large data sets in UI figures

In figures created with the `uifigure` function, the graphics rendering performance on large data sets is improved in some cases, such as when displaying these types of plots:

- Surface plots of data larger than a 300-by-300 matrix
- Scatter plots with 20,000 markers or more
- Images where the data stored in the `CData` property is greater than 1 KB

In cases like these, the larger the data set, the greater the performance improvement becomes. These examples show the improvements:

- This code creates the surface plot approximately 5.9x faster in R2020a:

```
function timingTestSurface
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
ax = uiaxes(fig);
drawnow

for k=1:num
    tic
    s = surface(ax,peaks(500));
    s.EdgeColor = 'none';
    drawnow
    tocTimes(k) = toc;
    delete(s)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

**R2019b:** 3.019 s

**R2020a:** 0.5102 s

- This code creates the scatter plot approximately 1.1x faster in R2020a:

```
function timingTestScatter
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
ax = uiaxes(fig);
drawnow
x = linspace(0,3*pi,20000);
y = cos(x) + rand(1,20000);

for k=1:num
    tic
    s = scatter(ax,x,y);
    drawnow
    tocTimes(k) = toc;
    delete(s)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

**R2019b:** 0.0952 s

**R2020a:** 0.0890 s

- This code displays a 650-by-600-by-3 truecolor image array (1.17 MB) approximately 1.5x faster in R2020a:

```
function timingTestImage
num = 10;
tocTimes = zeros(1,num);
C = imread('ngc6543a.jpg');
fig = uifigure;
ax = uiaxes(fig);
drawnow

for k=1:num
    tic
    im = image(ax,C);
    drawnow
    tocTimes(k) = toc;
    delete(im)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:



**R2019b:** 0.0903 s

**R2020a:** 0.0588 s

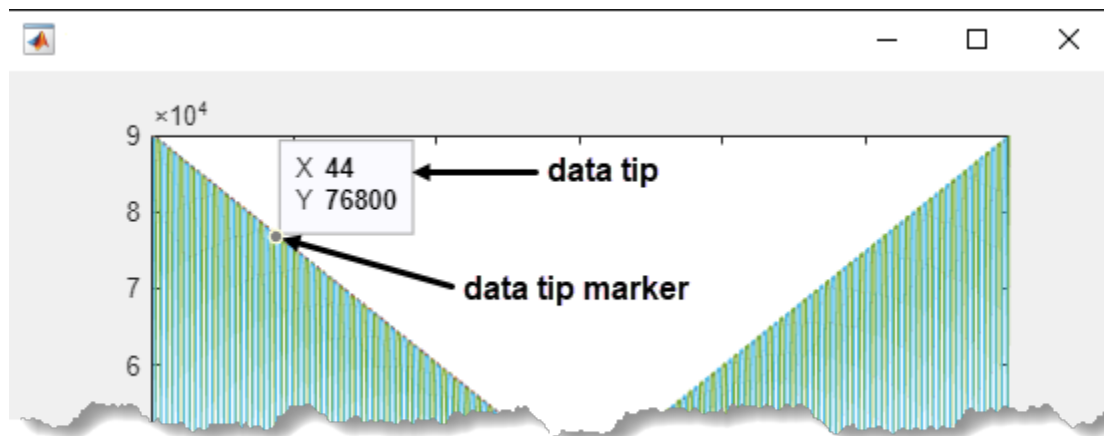
This code was timed on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card by calling the `timingTestSurface`, `timingTestScatter`, and `timingTestImage` functions.

## Data Tip Markers: Improved rendering performance of data tip markers in line plots of large data sets created in UI figures and MATLAB Online

In figures created with the `uifigure` function and in MATLAB Online, data tip markers for line plots of large data sets render faster and move more continuously in R2020a than in R2019b. This improvement can be seen when the axes are created with either the `axes` or `uiaxes` function.

For example, on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card, when you move your mouse quickly over the plot lines created by this code, the data tip marker moves more smoothly and tracks your mouse motion more closely in R2020a than in R2019b.

```
fig = uifigure;
ax = axes(fig);
plot(ax,magic(300));
```



## Icon Property: Improved rendering performance for buttons and tree nodes with icons

Creating `Button`, `ToggleButton`, and `TreeNode` objects with icons using the `Icon` property is significantly faster. The performance improvement gets better the larger the icon file size, or the more components with icons you have in your app.

For example, this code creates a button with an icon (file size 2.86 MB) approximately 4.6x faster in R2020a:

```
function timingTestButton
r = rand(1000,1000,3);
```

```

imwrite(r, 'testimage.png')
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
drawnow

for k = 1:num
    tic
    btn = uibutton(fig, 'Icon', 'testimage.png');
    drawnow
    tocTimes(k) = toc;
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end

```

The approximate execution times are:

**R2019b:** 0.705 s

**R2020a:** 0.154 s

This code was timed on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card by calling the `timingTestButton` function.

## Functionality being removed or changed

### bench Function: Problem sizes have increased for numerical computation tasks

#### *Behavior change*

Starting in R2020a, problem sizes have increased for the numerical computation tasks (LU, FFT, ODE, and Sparse) so that the ranking of machines using bench test results are not dominated by the 2-D and 3-D graphics tasks. In previous releases, the 2-D and 3-D tasks take significantly longer to complete compared to the numerical computation tasks and therefore contribute disproportionately to the ranking of machines.

This table shows different task execution times in R2020a using a *Windows 10, Intel Xeon W-2133 @ 3.60 GHz* test system. The measured values are expressed in seconds.

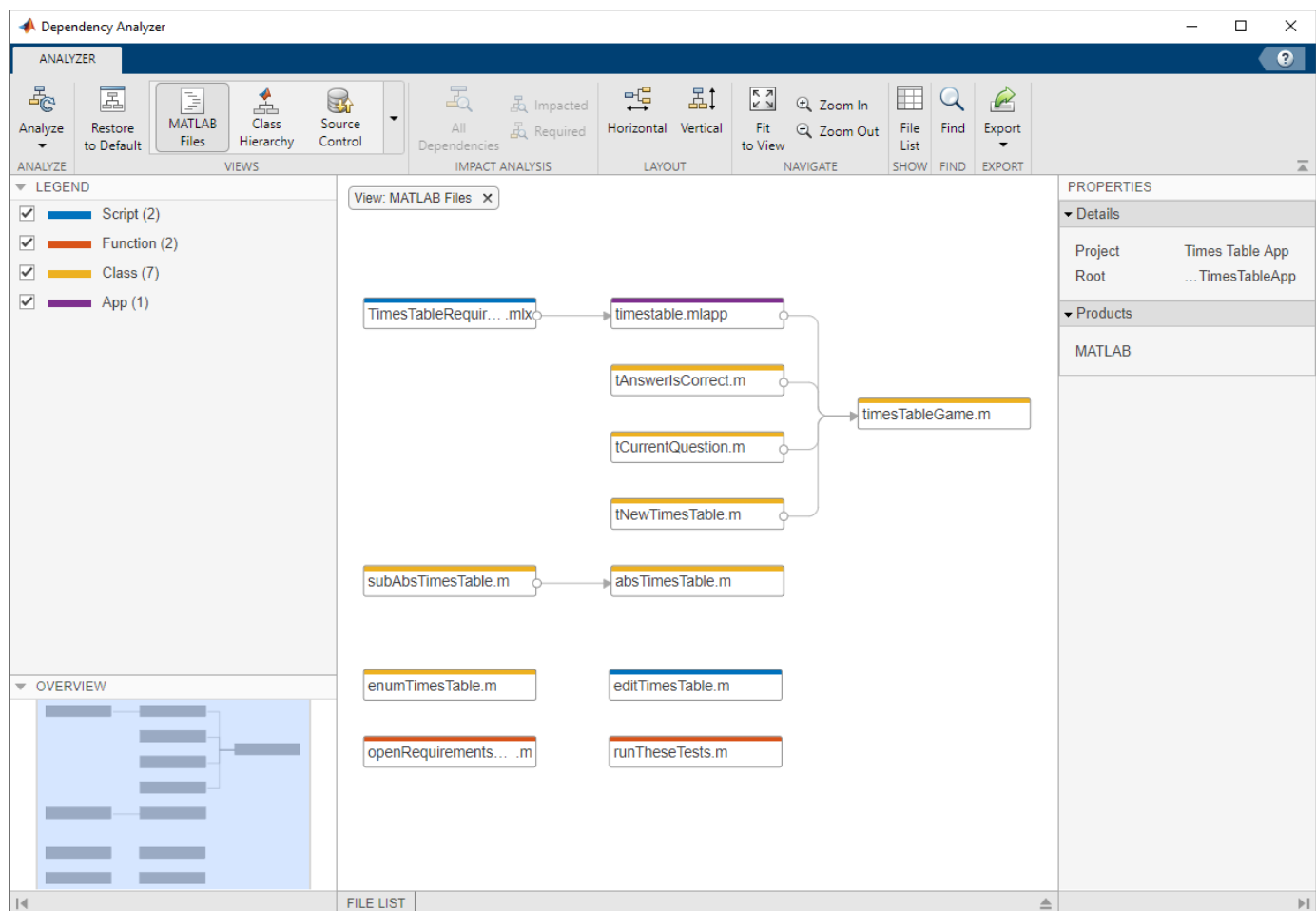
Test	New Problem Sizes	Old Problem Sizes
LU	0.388	0.068
FFT	0.312	0.112
ODE	0.480	0.015
Sparse	0.488	0.103
2-D	0.301	0.307
3-D	0.296	0.303

## Software Development Tools

### Dependency Analyzer: Improved navigation, filtering, and highlighting for project dependencies

In R2020a, use the new Dependency Analyzer to analyze project dependencies with improved navigation, filtering, and highlighting.

On the **Project** tab, in the **Tools** section, click **Dependency Analyzer**.



Run a dependency analysis to:

- Visualize the project structure and dependencies when you setup or explore a project for the first time. For more details, see [Explore the Dependency Graph, Views, and Filters](#).
- Find products and toolboxes required by your design. For more details, see [Find Required Products and Toolboxes](#).
- Investigate and resolve problems before sharing, packaging or submitting your project to source control. For more details, see [Investigate and Resolve Problems](#).
- Assess the impact of the changes you made on the rest of the design. For more details, see [Find File Dependencies](#).

For an example showing how to perform an impact analysis to find and run the tests affected by modified files, see [Perform Impact Analysis with a Project \(Simulink\)](#).

### Behavior Change

Workflow	R2019b	R2020a
Examine project dependencies and problem files using the file list.	At the top right of the <b>Dependency Analysis</b> view, select <b>Table View</b> .  MATLAB switches to the file list view.	In the Dependency Analyzer, click <b>File List</b> .  MATLAB opens the file list in the same view as the graph. See <a href="#">Investigate and Resolve Problems</a> .
Investigate how two files are related and highlight where the dependency is introduced.	Expand the file in the graph by clicking the arrows next to the file name. To highlight the dependency, double-click the line number in the expanded file.	Select the dependency arrow. In the <b>Properties</b> pane, in the <b>Details</b> section, you can see the full paths of the files you are examining, the dependency type, and where the dependency is introduced.  To highlight where the dependency is introduced, in the <b>Details</b> section, click the link under <b>Impacted</b> . See <a href="#">Investigate Dependency Between Two Files</a> .
Color files in the dependency graph by type, source control status, or label.	On the <b>Dependency Analysis</b> tab, in the <b>Group By</b> menu, select the option you want.	In the Dependency Analyzer, in the <b>Views</b> section, use the different views to explore your project files dependencies. See <a href="#">Explore the Dependency Graph, Views, and Filters</a> .

### Project Checks: Run all project checks programmatically

You can now use `runChecks` to run all project checks programmatically.

### Project API: Get latest Git revision programmatically

You can now programmatically get the latest Git revision for every file in your project.

For a project under Git source control, use `currentProject` to create a project object from the currently loaded project.

```
proj = currentProject;
```

Get the latest Git revision of the project file number `fileNumber`.

```
proj.Files(fileNumber).Revision
```

For an example on how to get the latest revisions for modified files, see [List Modified Files in Project](#).

## Unit Testing Framework: Add custom details to TestResult objects

Starting in R2020a, you can add data to the `Details` property of `TestResult` objects when you create your plugins. To append a field to the `Details` structure, use the `append` method of the `matlab.unittest.plugins.plugindata.ResultDetails` class. For more information, see [Write Plugin to Add Data to Test Results](#).

## Unit Testing Framework: Assert that test session ran with no failure

The `matlab.unittest.TestResult` class has a new method `assertSuccess`, which enables you to assert that no failing conditions were encountered during a test session. For example, run the tests defined in `MyTestClass` and assert that none of them failed.

```
result = assertSuccess(runtests('MyTestClass'));
```

## Unit Testing Framework: Run tests from the Live Editor toolstrip

You can now run tests from the MATLAB Live Editor toolstrip. When you open a function-based test file with a `.mlx` extension, the Live Editor toolstrip has options to:

- Run all tests in the file.
- Run the test at your cursor location.

You can customize the test run with options, such as running tests in parallel (which requires Parallel Computing Toolbox) or running tests with a specified level of output detail.

## Unit Testing Framework: Generate test reports including test tags

Starting in R2020a, test reports generated using the `TestReportPlugin` class display the test tags for tagged test suite elements. You can generate tagged test reports in DOCX, HTML, and PDF formats. For information about test tags, see [Tag Unit Tests](#).

## App Testing Framework: Perform press gestures with different selection types

The App testing framework supports mouse selection types in press gestures that are performed on UI figures. To specify the selection type, use the `'SelectionType'` name-value pair argument. For example:

```
f = uifigure;
testCase = matlab.uitest.TestCase.forInteractiveUse;
testCase.press(f, 'SelectionType', 'open')
```

## Mocking Framework: Add events to mock objects

When creating a mock object, you can add events to the object in addition to properties and methods. To specify the events to mock, use the `createMock` method with the `'AddedEvents'` name-value pair argument. To add events to the mock, the mock object must derive from a handle class. For example:

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock, behavior] = testCase.createMock(?handle, 'AddedEvents', {'EventA', 'EventB'});
```

## Mocking Framework: Specify when framework should do nothing

You can specify that the unit testing framework should do nothing each time a mock object method is called or a mock object property is set. Define this behavior with the `matlab.mock.actions.DoNothing` class.

For example, create a mock object with a property `PropA`. Define behavior such that the property is unchanged when it is assigned a value of 0:

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behavior] = testCase.createMock('AddedProperties','PropA');

import matlab.mock.actions.DoNothing
when(behavior.PropA.setToValue(0),DoNothing);
```

Use the mock to assign a nonzero value to `PropA`.

```
mock.PropA = 5

mock =

    Mock with properties:

        PropA: 5
```

Assign a value of 0 to the property. Due to the defined mock object behavior, the value of `PropA` remains unchanged.

```
mock.PropA = 0

mock =

    Mock with properties:

        PropA: 5
```

## Functionality being removed or changed

### ProfileReport will be removed in a future release

#### Warns

The `ProfileReport` class will be removed in a future release. Use `CoverageReport` or `CoberturaFormat` instead. Unlike `ProfileReport`, which supports running tests only in serial mode, these two classes can be used to generate reports when tests run in either serial or parallel mode.

To update your code, change instances of `ProfileReport` to `CoverageReport` or `CoberturaFormat`. This table shows an example of how you can update your code.

Before	After
<pre>import matlab.unittest.TestRunner import matlab.unittest.plugins.CodeCoveragePlugin import matlab.unittest.plugins.codecoverage.CoverageReport  runner = TestRunner.withNoPlugins; plugin = CodeCoveragePlugin.forFolder('myTests', ...     'Producing',ProfileReport); runner.addPlugin(plugin)</pre>	<pre>import matlab.unittest.TestRunner import matlab.unittest.plugins.CodeCoveragePlugin import matlab.unittest.plugins.codecoverage.CoverageReport  runner = TestRunner.withNoPlugins; plugin = CodeCoveragePlugin.forFolder('myTests', ...     'Producing',CoverageReport); runner.addPlugin(plugin)</pre>

To create a MATLAB Profiler Coverage Report without specifying a `ProfileReport` format, see [Determine Code Coverage Using the Profiler](#).

## External Language Interfaces

### C++ Interface: MATLAB data type for C++ array and `std::vector`

MATLAB provides an interface, `clib.array`, which wraps C++ native arrays and `std::vector` types. To create an array of C++ array objects, call the MATLAB `clibArray` function. To convert a MATLAB array to a C++ array, call `clibConvertArray`. For more information, see [MATLAB Object For C++ Arrays](#).

For information about creating a MATLAB interface, see [Build MATLAB Interface to C++ Library](#).

### Version History

By default, beginning in R2020a, MATLAB returns a `clib.array` object instead of the equivalent MATLAB array for primitive types.

To continue to treat a return argument as a MATLAB array, call `clibgen.generateLibraryDefinition` or `clibgen.buildInterface` with the 'ReturnCArrays' argument set to `false`.

To update your code to use `clib` arrays, note that when you rebuild the library the MATLAB type for these output arguments changes to `clib.array.libname.classname`. Also, MATLAB automatically defines more parameters. In general, you still need to provide SHAPE information.

### C++ Interface: Supported data types

Functionality in a C++ shared library using these types is included in the MATLAB interface to the library.

- `std::shared_ptr`
- Pointer and array data members.
- Double pointers (\*\*) to custom classes used as function or method parameter types. Double pointers to primitive types are not supported.

These types are equivalent to MATLAB `char`:

- `wchar_t`
- `char16_t`
- `char32_t`

These types are equivalent to MATLAB `string`:

- `char *`
- `std::wstring`
- `std::u16string`
- `std::u32string`

For more information, see [MATLAB to C++ Data Type Mapping](#). To determine if it is possible to publish an interface to your library, see [Limitations to C/C++ Support](#).

## C++ Interface: Lifetime management of C++ objects

If a library creates an object, then the library is responsible for releasing the memory. Likewise, if MATLAB creates the object, then MATLAB is responsible for releasing the memory. MATLAB lets you control the lifetime management of objects by specifying 'ReleaseOnCall' and 'DeleteFcn' arguments in the library definition file. For more information, see Lifetime Management of C++ Objects in MATLAB.

## MATLAB Data Array: Support for N-D row-major memory layout

You can create an `N-D matlab::data::Array` object with row-major memory layout. Previously, the `createArrayFromBuffer` function created row-major arrays only in 2-D. To create a `matlab::data::Array` object with row-major memory layout, set the `createArrayFromBuffer` argument `memoryLayout` to `MemoryLayout::ROW_MAJOR`. To determine the memory layout for an existing `matlab::data::Array` object, call `getMemoryLayout`.

## MATLAB COM Server: Register MATLAB without administrative privileges

If you do not have an administrator account on a Microsoft Windows system or you do not start MATLAB with administrative privileges, you can use the `comserver` function to register MATLAB as a COM server for your user account. If you have an administrator account, then you also can call `comserver` to register MATLAB for all users. For more information, see Register MATLAB as COM Server. For general information about the MATLAB COM Automation server, see Calling MATLAB as COM Automation Server.

## Java interface: MATLAB support for OpenJDK™ 8 (Hot Spot)

As of R2020a, MATLAB supports OpenJDK 8 (Hot Spot) from <https://adoptopenjdk.net/>.

## Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2020 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2020 for Fortran	macOS
Added	Apple Xcode 11.x	macOS
Discontinued	Intel Parallel Studio XE 2017	Windows macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

## Functionality being removed or changed

**`createSoapMessage`, `callSoapService`, and `parseSoapResponse` will be removed**  
Warns



Consider using `matlab.wsdL.createWSDLClient` instead of the `createSoapMessage`, `callSoapService`, and `parseSoapResponse` functions to communicate with Web services using Simple Object Access Protocol (SOAP). There is no direct function replacement for the SOAP functions, but when you create a WSDL interface, you have access to the Web service functionality.

### **createClassFromWsdL will be removed**

#### *Warns*

The `matlab.wsdL.createWSDLClient` function replaces the `createClassFromWsdL` function to communicate with Web services from MATLAB using Web Services Description Language (WSDL). `matlab.wsdL.createWSDLClient` enables you to specify additional information needed to access the WSDL document. For more information, see `weboptions`.

To get started using `matlab.wsdL.createWSDLClient`, follow these steps.

- 1 Download supported versions of the Java JDK and Apache CXF programs. For more information, see `Set Up WSDL Tools`.
- 2 Set the paths to these programs:

```
matlab.wsdL.setWSDLToolPath('JDK',jdk,'CXF',cxf)
```

where `jdk` is the path to the JDK installation and `cxf` is the path to the CXF program.

To update your code, replace calls to `createClassFromWsdL` with calls to `matlab.wsdL.createWSDLClient`. For example, for a Web service with this URL:

```
url = 'https://examplesite.com/samplewebservice';
```

replace this call to `createClassFromWsdL`:

```
createClassFromWsdL(strcat(url,'?WSDL'))
```

with:

```
matlab.wsdL.createWSDLClient(url)
```

---

**Note** `matlab.wsdL.createWSDLClient` does not support RPC-encoded WSDL documents.

---

## Hardware Support

### MATLAB Support Package for Ryze Tello Drones: Control Ryze Tello drone from MATLAB and acquire sensor and image data

The MATLAB Support Package for Ryze® Tello Drones is available from release R2020a onwards.

The support package includes functions to pilot Ryze Tello and Ryze Tello EDU drones by sending MATLAB commands to control its direction, speed, and orientation. You can also read flight navigation data such as speed, height, and orientation, capture images, and stream live video into MATLAB from the drone's first-person view (FPV) camera.

### Support added for Raspberry Pi 4B model board

You can use the MATLAB Support Package for Raspberry Pi Hardware with the Raspberry Pi 4B board.

### Deploy deep learning applications on Raspberry Pi hardware

The MATLAB Support Package for Raspberry Pi Hardware now enables you to deploy deep learning applications on the hardware. The deep learning applications continue to run on the Raspberry Pi even if the hardware is disconnected from the computer. To support deployment, the Raspberry Pi functions listed in Functions Supported for Deployment (MATLAB Support Package for Raspberry Pi Hardware) are enhanced to generate C++ code. Like any MATLAB function, you can deploy the deep learning application using these steps in Workflow to Deploy MATLAB Function on Raspberry Pi (MATLAB Support Package for Raspberry Pi Hardware) with an additional step of creating a deep learning configuration object and assigning it to the `CoderConfig.DeepLearningConfig` property of Raspberry Pi before deployment.

For example, to deploy the deep learning application `raspi_webcam_resnet.m`:

- 1 Create a configuration object for the Raspberry Pi hardware and set the language of the generated code to C++ using `TargetLang`.

```
t = targetHardware('Raspberry Pi');  
t.CoderConfig.TargetLang = 'C++';
```

- 2 Create an `arm-compute` deep learning configuration object and assign it to the `DeepLearningConfig` property of the Raspberry Pi configuration object.

```
dlcfg = coder.DeepLearningConfig('arm-compute')  
dlcfg.ArmArchitecture = 'armv7';  
t.CoderConfig.DeepLearningConfig = dlcfg;
```

- 3 Deploy the application on the Raspberry Pi hardware.

```
deploy(t, 'raspi_webcam_resnet.m')
```

### Read GPS Data from GPS Receiver Connected to Arduino Hardware

The MATLAB Support Package for Arduino Hardware enables you to read GPS data from the GPS receiver connected to an Arduino hardware.

## Use BNO055 Sensor with Sensor Fusion and Tracking Toolbox, and Navigation Toolbox to Estimate Orientation

You can read acceleration, angular velocity, and magnetic field up to 200Hz in the non-fusion mode of the BNO055 IMU sensor connected to the Arduino hardware. To estimate orientation, you can use the sensor with Sensor Fusion and Tracking Toolbox™, and Navigation Toolbox™.

## Enable Code Generation of MATLAB Arduino Functions Inside a MATLAB Function Block for I2C and SPI

In addition to the existing support of ADC, PWM, and Digital Read/Write, you can now generate code for Arduino MATLAB functions inside a MATLAB function block for I2C and SPI.

## Functionality being changed or removed

### The `i2cdev` and `spidev` functions will be removed in R2020a

*Warns*

Use the device instead of `i2cdev` and `spidev` to connect to I2C or SPI devices on Arduino hardware .

### The property `Pins` of `servo` object will be removed in R2020a

*Warns*

Use the property `Pin` instead of `Pins` to get the pin number of the Arduino hardware and the Adafruit® Motor Shield V2 for Arduino hardware to which the servo motor is connected. For more information, see [Connection to servo motor on Arduino](#) and [Connection to servo motor on Adafruit Motor Shield V2](#) .

### The class `arduinoio.LibraryBase` will be removed in R2020a

*Warns*

Use the class `matlabshared.addon.LibraryBase` instead of `arduinoio.LibraryBase` for deriving Arduino add-on libraries.

### MATLAB support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in R2020a

*Warns*

The support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in R2020a.



# R2019b

---

**Version: 9.7**

**New Features**


**Bug Fixes**

**Version History**

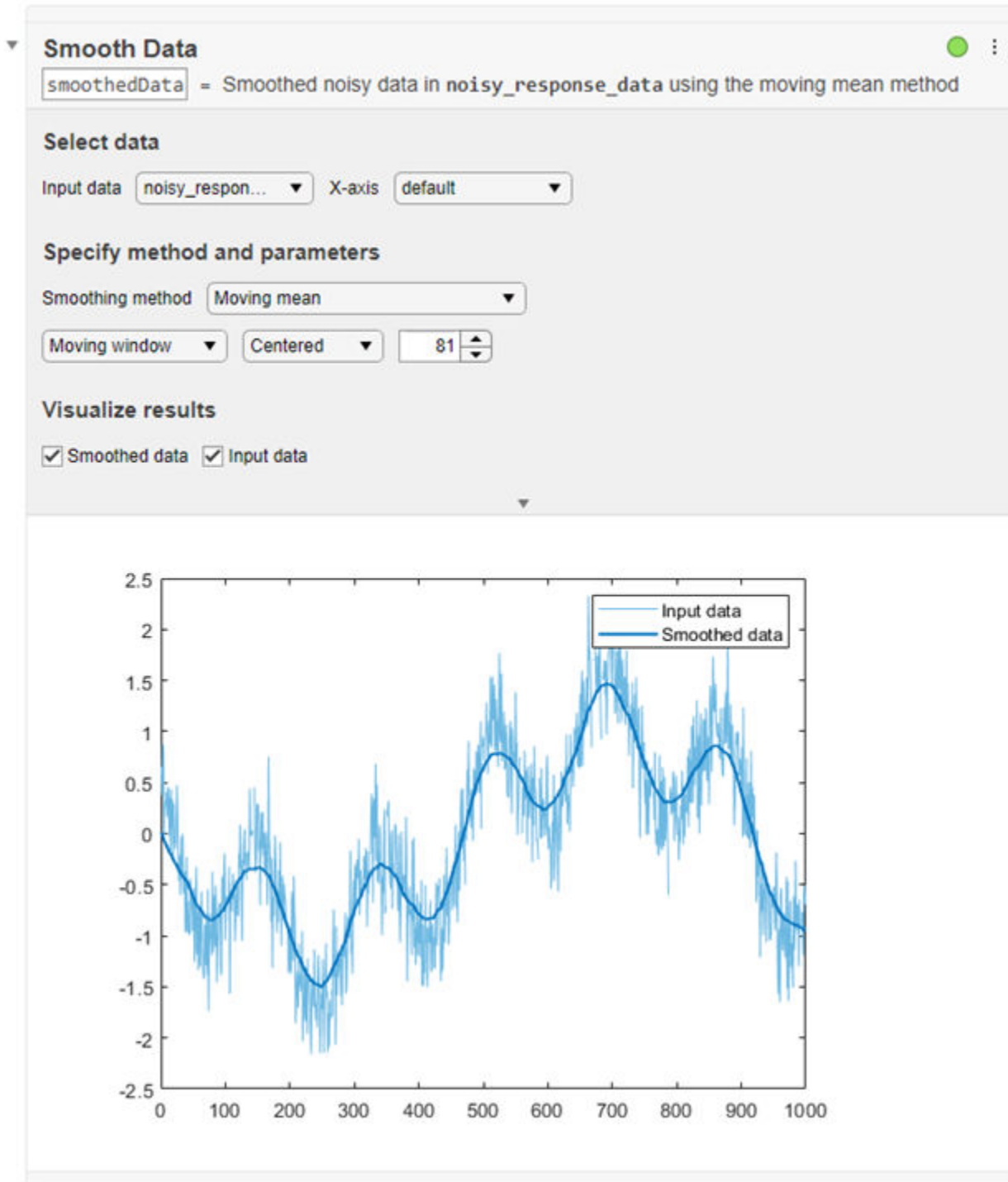
## Environment

### **Live Editor Tasks: Add interactive tasks to live scripts to explore parameters and automatically generate code**

Live Editor tasks are app-like interfaces that can be added to a live script to perform a specific set of operations. Use tasks to reduce development time, errors, and time spent plotting. Tasks automatically generate code that becomes part of your live script.

To add a task to a live script, go to the **Live Editor** tab, click  **Task** ▾, and select from the available tasks. You also can type the name of the task in a live script code block. As you type, the Live Editor displays possible matches, and you can select and insert the desired task.

For example, add the **Smooth Data** task to a live script to smooth noisy data.

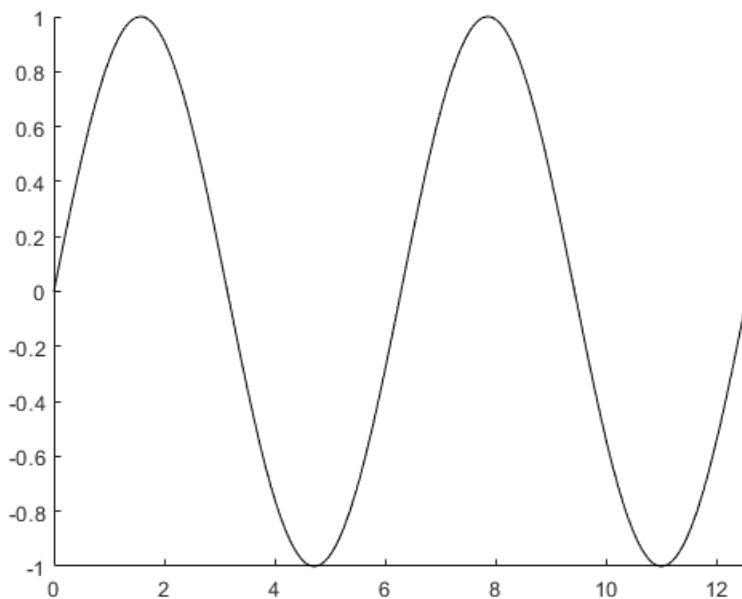


For more information, see [Add Interactive Tasks to a Live Script](#).

## Live Editor Output: Animate plots to show changes in data over time

You can include for-loop animations in the Live Editor to show changes in plotted data over time. For example, this code animates a line growing as it accumulates 2,000 data points in the Live Editor. The `drawnow` function displays the changes after each iteration through the loop.

```
h = animatedline;  
axis([0 4*pi -1 1])  
x = linspace(0,4*pi,2000);  
  
for k = 1:length(x)  
    y = sin(x(k));  
    addpoints(h,x(k),y);  
    drawnow  
end
```



## Live Editor Output: Adjust the width of columns in tables

Adjust the width of table columns in the Live Editor by clicking and dragging the column border to the desired width.

## Live Editor Output: Scroll through and copy data in arrays such as cell arrays, object arrays, and struct arrays

You can use scroll bars to explore data in cell arrays, object arrays, and struct arrays. You also can scroll through data in string arrays, numeric arrays, categorical arrays, `datetime` arrays, duration arrays, and `calendarDuration` arrays.

To copy individual data values in output, select a value, right-click it, and choose the **Copy Selection** option.



Patients = 14x5 cell

	1	2	3	4	5
1	'Gender'	'Age'	'Height'	'Weight'	'Smoker'
2	'M'	38	71	176	1
3	'M'	43	69	163	0
4	'M'	38	64	121	0
5	'M'	36	64	121	1
6	'M'	42	64	142	0
7	'M'	41	64	129	0
8	'F'	38	64	131	0
9	'F'	40	67	133	1

## Live Editor Export: Customize figure format as well as document paper size, orientation, and margins when exporting

You can programmatically change the resolution and format of images when exporting to PDF and LaTeX in the Live Editor using settings. For example, this code specifies the JPEG figure format and a resolution of 1200 DPI when exporting in the Live Editor for the current session.

```
s = settings;
s.matlab.editor.export.FigureFormat.TemporaryValue = 'jpeg';
s.matlab.editor.export.FigureResolution.TemporaryValue = 1200;
```

You also can programmatically change the paper size, orientation, and margins when exporting to PDF, Microsoft Word documents, and LaTeX in the Live Editor. For example, this code specifies the Legal paper size and landscape page orientation when exporting in the Live Editor for the current MATLAB session.

```
s.matlab.editor.export.pagesetup.PaperSize.TemporaryValue = 'Legal';
s.matlab.editor.export.pagesetup.Orientation.TemporaryValue = 'Landscape';
```

To change the figure format, document paper size, orientation, and margins for an individual export document type, specify the setting for that individual document type. For example, this code specifies a portrait page orientation when exporting to PDF documents in the Live Editor, and a landscape page orientation when exporting to all other document types. Set the personal value instead of the temporary value for the settings to ensure that the values persist across MATLAB sessions.

```
s.matlab.editor.export.pagesetup.pdf.Orientation.PersonalValue = 'Portrait';
s.matlab.editor.export.pagesetup.Orientation.PersonalValue = 'Landscape';
```

For more information, see `matlab.editor` Settings.

## Live Editor Code: Duplicate one or more lines of code

You can reuse one or more lines of code in a live script by duplicating them. To duplicate selected lines, right-click them and select **Duplicate Line(s)**. You also can use the keyboard shortcut **Ctrl+Shift+C** (or **Cmd+Shift+C** on macOS).

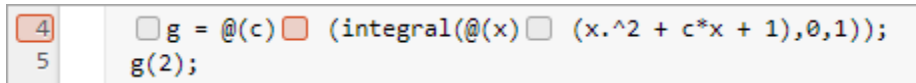
## Live Editor Code: Suppress Code Analyzer warning messages

To suppress Code Analyzer warnings for a single line in the Live Editor, right-click a warning and select **Suppress Message... > On This Line**. To suppress the warning for the entire document, select **Suppress Message... > In This File**.

Error messages cannot be suppressed.

## Live Editor Debugging: Set breakpoints for anonymous functions

In the Live Editor, you can set breakpoints for anonymous functions. To add a breakpoint, click the gray area to the left of an executable line that contains the anonymous function. MATLAB adds a breakpoint to the line and a disabled breakpoint to the left of each anonymous function in the line. Click the disabled breakpoint to enable it.



## Live Editor Internationalization: Add non-English language such as Chinese, Japanese, and Korean characters on Windows and macOS Platforms

You can specify different input methods on Windows and macOS systems to add non-English languages such as Chinese, Japanese, and Korean characters to code and text in the Live Editor.

## Add-On Manager: Update MATLAB and other installed add-ons

You can update MATLAB and other installed add-ons to their latest version in the Add-On Manager. To view and install available updates, on the **Home** tab, click **Help > Check for Updates**. The Add-On Manager opens. Select the **Update** tab to view available updates. Click the **Update** button to the right of an available update to install it. For more information, see [Get and Manage Add-Ons](#).

## Add-On Manager: Programmatically manage add-ons by name

Enable, disable, and uninstall add-ons programmatically by name using the `matlab.addons.enableAddon`, `matlab.addons.disableAddon`, and `matlab.addons.uninstall` functions.

For example, suppose that you have an add-on called `Random File Name Creator` installed on your system. Disable `Random File Name Creator` by name.

```
matlab.addons.disableAddon("Random File Name Creator")
addons = matlab.addons.installedAddons
```

```
addons =
```

```
1x4 table
```

Name	Version	Enabled	Identifier
"Random File Name Creator"	"1.0"	false	"75442144-f751-4011-bm0e-32b6fb2f1433"

## Settings: Create persistent settings for custom apps, toolboxes, and across MATLAB sessions

You can add your own custom settings in MATLAB to store and access data programmatically across a single or multiple MATLAB sessions.

For example, create the settings group `mysettings` and the setting `MyWorkAddress` inside it.

```
s = settings;
addGroup(s,'mysettings','Hidden',false);
addSetting(s.mysettings,'MyWorkAddress','Hidden',false);
s.mysettings.MyWorkAddress.PersonalValue = '3 Apple Hill Drive';
```

You can then use the setting value programmatically in your code.

```
fprintf("I work at %s.\n", s.mysettings.MyWorkAddress.ActiveValue)
```

```
I work at 3 Apple Hill Drive.
```

For more information, see [Create Custom Settings](#). To create settings for custom toolboxes, see [Create Factory Settings for Toolboxes](#).

## MATLAB Drive: Share folders and collaborate with others

If you have MATLAB Drive Connector installed and actively syncing, you can share folders in your MATLAB Drive directly from MATLAB. To share a folder, right-click the folder in the Current Folder browser and select **Share**.

You can share folders with a view-only link, or invite individual collaborators and set their editing permissions. Invitations can be accepted or declined. After a folder is shared, you can manage the permissions of invited members, rescind invitations, or send additional invitations at any time.

For more information, see [Share Folders in MATLAB](#).

## Functionality being removed or changed

### Live editor animations enabled by default

*Behavior change*

Starting in R2019b, for-loop animations in the Live Editor are enabled by default. To disable animations in the Live Editor, set the `matlab.editor.AllowFigureAnimation` setting to `false`:

```
s = settings;
s.matlab.editor.AllowFigureAnimation.PersonalValue = false;
```

## Language and Programming

### **size Function: Find lengths of multiple array dimensions at a time**

You can now use a vector dimension argument to query multiple array dimension lengths at a time with the `size` function. For example, `sz = size(A, [1 3])` returns a 1-by-2 row vector containing the lengths of dimensions 1 and 3 of the input array `A`.

### **matches Function: Determine if input strings are equal**

You can determine if two input strings are equal. For more information, see `matches`.

### **Hexadecimal and Binary Numbers: Specify numbers using hexadecimal and binary literals**

You can write numbers using hexadecimal and binary notation. For example, you can write the number 42 as `A = 0x2A` using the `0x` prefix to indicate hexadecimal format. For more information, see `Hexadecimal and Binary Values`.

### **Indexing: Use dot indexing into function calls**

You can now use dot indexing to index into the result of a function call. MATLAB evaluates the function and then applies the dot indexing operation to the result.

For example, this function creates a structure:

```
function out = createStruct(in)
out = struct("aField", in);
end
```

You can call this function and immediately access the structure field it creates:

```
createStruct(3).aField
```

For more information, see `Indexing into Function Call Results`.

### **System object authoring improvements: Property validation support and simplified class inheritance**

When you author System objects, you can now use property validation to restrict property values. For information about property validation, see `Validate Property Values`.

The methods from the `matlab.system.mixin.CustomIcon`, `matlab.system.mixin.Nondirect`, `matlab.system.mixin.Propagates`, and `matlab.system.mixin.SampleTime` mixin classes are now directly included with `matlab.System`. You no longer need to inherit from these mixin classes when authoring System objects.

## Function Input Arguments: Declare function input arguments to restrict values

Function argument validation is a way to declare specific restrictions on function input arguments. Using function argument validation, you can constrain the class, size, and other aspects of function input values without writing code in the body of the function to perform these tests. For more information, see [Function Argument Validation](#).

## namedargs2cell Function: Convert structure containing name-value pairs to cell array

Convert a scalar structure array that contains name-value data into an interleaved cell array suitable for passing to functions that accept name-value pair cell arrays. This function is typically used with the name-value structure created using function argument validation. For more information, see [namedargs2cell](#).

## delete, dir,.isfile, isfolder, and what Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage

You can now use the `delete`, `dir`, `isfile`, `isfolder`, and `what` functions to work with remote files and folders. To access remote locations, you must specify the full path using a uniform resource locator (URL). For example, check if a file exists on the specified path in Amazon S3 Cloud.

```
result = isfile('s3://bucketname/path_to_file/my_image.jpg')
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## Suggested Corrections: Correct errors with two new classes

In addition to the `AppendArgumentsCorrection` class, you can now provide a suggested fix using the `ReplaceIdentifierCorrection` and `ConvertToFunctionNotationCorrection` classes.

- To suggest replacing an identifier in the function call where the exception was thrown, use the `ReplaceIdentifierCorrection` class.
- To suggest the function notation equivalent of the dot notation expression from which the exception was thrown, use the `ConvertToFunctionNotationCorrection` class.

## error Function: Provide suggested fix for uncaught exception

You can now use the `error` function to provide a suggested fix when the thrown exception is not caught. To use this functionality, specify the first argument of `error` as an object that suggests the correction. For example, display a suggestion for missing arguments by specifying an `AppendArgumentsCorrection` instance.

```
aac = matlab.lang.correction.AppendArgumentsCorrection(["InitialCondition", "0"]);
error(aac, msgID, msgText)
```

## Functionality being removed or changed

### Program files larger than 128 MB or with high complexity produce error

*Behavior change*

Starting in R2019b, program files larger than approximately 128 MB do not open or run. For files that contain only code (for example, `.m` and `.p` files), this limit affects the file size. For files that store more than just code (for example, `.mlx` files), it affects the size of the code. Running statements larger than 128 MB, either directly in the Command Window or using the `eval` function, also is not supported. In addition, code with high levels of complexity, such as a large number of deeply nested `if` statements, is not supported.

Large program file or statement sizes could result in unpredictable behavior and often occurred when using large portions of code (for example, over 500 lines) to define variables with constant values. To decrease the size of program files, consider defining the variables and saving them in a data file (for example, a MAT-file or `.csv` file). Then you can load the variables instead of executing code to generate them. This adjustment not only decreases the file size of your program but also can increase performance.

### Change in rules for function precedence order

#### *Behavior change*

Starting in R2019b, MATLAB changes the rules for name resolution, impacting the precedence order of variables, nested functions, local functions, and external functions. These rules are described in [Function Precedence Order](#). For information about the changes and tips for updating your code, see [Update Code for R2019b Changes to Function Precedence Order](#).

- Identifiers cannot be used for two purposes inside a function.
- Identifiers without explicit declarations might not be treated as variables.
- Variables cannot be implicitly shared between parent and nested functions.
- Change in precedence of compound name resolution.
- Anonymous functions can include resolved and unresolved identifiers.

The behavior of the `import` function has changed.

- Change in precedence of wildcard-based imports
- Fully qualified import functions cannot have the same name as nested functions
- Fully qualified imports shadow outer scope definitions of the same name.
- Error handling when import is not found.
- Nested functions inherit import statements from parent functions.

### Some repetition arguments for `repmat` function now produce errors

#### *Behavior change*

Starting in R2019b, some `repmat` syntaxes involving nonscalar or empty repetition arguments will produce an error. The following table describes how to update these syntaxes.

Errors	Use Instead
<code>repmat(A, r1, r2)</code> , where <code>r1</code> and <code>r2</code> are row vectors	<code>repmat(A, [r1 r2])</code>
<code>repmat(A, empt)</code> , where <code>empt</code> is an empty array	<code>repmat(A, 1)</code>
<code>repmat(A, empt1, empt2)</code> , where <code>empt1</code> and <code>empt2</code> are empty arrays	<code>repmat(A, 1)</code>

Errors	Use Instead
<code>repmat(A,n,empt)</code> , where <code>n</code> is an integer and <code>empt</code> is an empty array	<code>repmat(A,[n 1])</code>

### Noninteger or complex dimension order arguments for permute function now produce errors

*Behavior change*

Starting in R2019b, the `permute` syntax `permute(A,dimorder)` produces an error when `dimorder` is a noninteger or complex value. Instead, specify real, positive integer values for `dimorder`.

### Compare scalar enumerations in cell arrays to text using the strcmp function

*Behavior change*

`strcmp` now returns logical 0 or 1 for comparisons of scalar enumerations in cell arrays to text. Previously, `strcmp` always returned 0.

For example, compare a `matlab.lang.OnOffSwitchState` enumeration in a cell and a character vector.

```
enum = matlab.lang.OnOffSwitchState.off;
TF = strcmp({enum}, 'off')
```

TF =

```
logical
    1
```

In previous releases, this comparison using `strcmp` returned 0.

You can also compare `{enum}` to a cell array of character vectors or to a string array.

```
TF = strcmp({enum}, {'on', 'off'})
```

TF =

```
1x2 logical array
    0    1
```

However, comparison to strings in cell arrays, such as `strcmp({enum}, {"off"})`, always returns 0. Cell arrays of string arrays are not recommended.

### Behavior of nargin for functions that use function argument validation

*Behavior change*

If the function name input argument to `nargin` refers to a function that uses function argument validation, then the returned value is the number of positional arguments on the function line. This change in behavior can affect code that uses `nargin` with functions that are subsequently updated to use the function argument validation feature. For more information, see `nargin` in Argument Validation.

### Dot-parenthesis syntax will not be allowed in validation function calls

*Behavior change in future release*

Calling validation functions from `properties` or `arguments` blocks using dot-parenthesis syntax in the function name will be disallowed in a future release. For example, this syntax will be invalid:

```
classdef MyClass
    properties
        A {a.('b').mustBePositive} % Dot-paren not allowed
    end
end
```

### Validation function must refer to the property being validated

*Behavior change in future release*

Property validation functions that do not refer to the property being validated will be disallowed in a future release. For example, in this class definition, the input to the `mustBePositive` function is the constant `10` instead of the property value.

```
classdef MyClass
    properties
        A {mustBePositive(10)} % Property not validated
    end
end
```

To validate the property value, either pass the property name explicitly, or allow MATLAB to pass the property implicitly:

```
classdef MyClass
    properties
        A {mustBePositive}
    end
end
```

### Function handles will not be allowed as inputs to property validation functions

*Behavior change in future release*

Passing function handles to property validation functions will be disallowed in a future release. If you define a validation function that must use a function handle, then define the function handle in the body of the validation function.

For example, it will be an error to pass the function handle `@sin` to the `customValFcn` function:

```
classdef MyClass
    properties
        A {customValFcn(A,@sin)} % Will result in error
    end
end

function customValFcn(A,fh)
    % function body
end
```

Instead, define the function handle in the custom validation function.

```
classdef MyClass
    properties
        A {customValFcn(A)}
    end
end
```



```
function customValFcn(A)
    fh = @sin;
    % function body
end
```

## Data Analysis

### Live Editor Tasks: Interactively preprocess data and generate code

Use Live Editor tasks to join tables; smooth data; handle outliers and missing data; remove trends; and find local minima, local maxima, and change points. Interactively explore the effects of your changes using generated plots. The tasks also automatically generate code that becomes part of your live script.

MATLAB now offers seven data preprocessing tasks:

- **Join Tables** — Combine two tables using key variables.
- **Smooth Data** — Smooth noisy data.
- **Clean Outlier Data** — Find, fill, or remove outliers.
- **Clean Missing Data** — Find, fill, or remove missing data.
- **Remove Trends** — Remove polynomial trend from data.
- **Find Local Extrema** — Find local maxima and minima.
- **Find Change Points** — Find abrupt changes in data.

To open tasks in the Live Editor, use the **Task** menu on the **Live Editor** tab. For more information, see [Add Interactive Tasks to a Live Script](#).

### groupfilter Function: Filter data in a table, timetable, or matrix by group

The `groupfilter` function allows you to filter rows of data by group according to a specified filtering method. For example, given a table `T`, group by variable `Var1` and return only the rows of `T` whose group has more than one element.

```
>> T = table(["A" "B" "C" "A" "A"],[1 2 3 4 5])
```

```
T =
```

```
5x2 table
```

Var1	Var2
"A"	1
"B"	2
"C"	3
"A"	4
"A"	5

```
>> G = groupfilter(T,'Var1',@(x) numel(x) > 1)
```

```
G =
```

```
3x2 table
```

Var1	Var2
"A"	4
"A"	5

```
"A"      1
"A"      4
"A"      5
```

## datetime Data Type: Detect formats with fractional seconds when converting text that represents dates and times

The `datetime` function allows you to convert text that includes fractional seconds without specifying the `'InputFormat'` name-value pair argument. For example, given text representing a date and time that includes milliseconds, convert the text to a `datetime` value.

```
d = datetime('2019-07-01 12:34:56.123')
```

```
d =
```

```
datetime
    01-Jul-2019 12:34:56
```

While the fractional seconds are detected, the default format for displaying values does not include fractional seconds. To display the fractional seconds, add `'.SSS'` to the `Format` property of `d`.

```
d.Format = 'dd-MMM-uuuu HH:mm:ss.SSS'
```

```
d =
```

```
datetime
    01-Jul-2019 12:34:56.123
```

For more information, see the `Format` property of `datetime`.

## table and timetable Data Types: Variable names can have any characters, including spaces and non-ASCII characters

Starting in R2019b, the names of variables in tables and timetables can have any Unicode characters, including spaces and non-ASCII characters. Previously, table and timetable variable names had to be valid MATLAB identifiers. For more information, see `table`, `timetable`, or `Access Data in Tables`.

This feature has several implications.

- While table and timetable variable names can have any characters, the `table2struct` and `summary` functions do modify variable names that are invalid when you use them as structure field names. The names of fields in a structure must be valid MATLAB identifiers.

To determine if a name is a valid identifier, use the `isvarname` function.

- You can access any table or timetable variable using dot notation. However, if a variable name is not a valid MATLAB identifier, then you must specify the name as an expression within parentheses following the dot. The expression can be a variable name enclosed in quotation marks, or a function that returns a character vector or string scalar.

For example, if the table `T` has variables named `'Age'` and `'Blood Pressure'`, then you can use dot notation to access the variables. You can use both of the following syntaxes to access `Age` because it is a valid MATLAB identifier.

```
T.Age
T.('Age')
```

However, to access the variable named 'Blood Pressure' using dot notation, you must use parentheses because the name is not a valid MATLAB identifier.

```
T.('Blood Pressure')
```

## Version History

- Starting in R2019b, table and timetable variable names with leading or trailing whitespace characters are not modified.

In previous releases, leading and trailing whitespace characters were deleted from variable names when you specified them using the 'VariableNames' name-value pair argument, or assigned them to the VariableNames property.

This change in behavior affects the `array2table`, `array2timetable`, `cell2table`, `table`, and `timetable` functions, and the VariableNames property of tables and timetables. To remove such characters manually, first use the `strtrim` function on the names, then assign them as variable names to the table or timetable.

- Starting in R2019b, MATLAB raises an error if you assign a table variable name that matches a dimension name, or a dimension name that matches a variable name. In previous releases, MATLAB raised a warning and modified the names so they were unique.

## tall Arrays: Operate on tall arrays with more functions, including `setdiff`, `xcorr`, and `outerjoin`

The functions listed in this table now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

<code>isregular</code>	<code>vartype</code>
<code>outerjoin</code>	<code>xcorr</code>
<code>setdiff</code>	<code>xcov</code>
<code>setxor</code>	

In addition, some functions have removed limitations with tall arrays.

Functions	Added Support
<code>min</code> and <code>max</code>	<code>min</code> and <code>max</code> now fully support tall arrays. <i>Previously, you could not specify multiple outputs.</i>
<code>conv</code> , <code>conv2</code> , and <code>convn</code>	<code>conv</code> , <code>conv2</code> , and <code>convn</code> now support more mixtures of tall arrays and in-memory arrays in the inputs. <i>Previously, the second input <i>B</i> could not be a tall array.</i>

Functions	Added Support
<code>innerjoin</code>	<p><code>innerjoin</code> now fully supports tall arrays.</p> <p><i>Previously, you could only specify one output argument, and one of the inputs was required to be an in-memory array.</i></p>
<code>intersect</code>	<p><code>intersect</code> now allows both inputs to be tall arrays.</p> <p><i>Previously, one of the inputs was required to be an in-memory array.</i></p>

### **tall Arrays: Avoid running out of memory due to temporary copies of data**

If you convert an in-memory array `A` into a tall array using `T = tall(A)`, then MATLAB avoids creating copies of the underlying data when you perform subsequent operations on `T`, which relaxes the memory requirements of many operations. This approach is useful if, for example, you have 8 GB of RAM and perform an operation on a 5 GB array that normally requires a temporary copy of the data.

Previously, the syntax `tall(A)` for an in-memory array `A` did not avoid creating temporary copies of data, and was bound by the same memory requirements as in-memory arrays.

## Data Import and Export

### **detectImportOptions Function: Specify the type of import options for delimited or fixed-width text files**

When detecting import options for a text file, the `detectImportOptions` function now enables you to specify the type of the text file as `'delimitedtext'` or `'fixedwidth'`. For more information, see `detectImportOptions`.

### **table and timetable Data Types: Read and write tabular data that has variable names containing any characters, including spaces and non-ASCII characters**

Starting in R2019b, you can read and write tables and timetables containing variable names that have any Unicode characters, including spaces and non-ASCII characters. Previously, table and timetable variable names had to be valid MATLAB identifiers.

To read tabular data that contains arbitrary variable names, such as variable names with spaces and non-ASCII characters, set the `PreserveVariableNames` parameter to `true`.

```
T = readtable('myFile.xls','PreserveVariableNames',true)
```

The following functions support reading and writing of tabular data containing variable names that contain any characters:

- **Import Tool**, `readtable`, `readtimetable`, and `parquetread`
- `detectImportOptions`, `spreadsheetImportOptions`, `delimitedTextImportOptions`, and `fixedWidthImportOptions`
- `tabularTextDatastore`, `spreadsheetDatastore`, and `parquetDatastore`
- `writetable`, `writetimetable`, and `parquetwrite`
- `save` and `load`

If you save a table or timetable to a MAT-file (using the `save` function), then the MAT-file contains both original and modified names for any variables whose names are not valid identifiers. If you load the MAT-file into R2019b or later (using the `load` function), then the table or timetable uses original variable names. If you load it into R2019a or earlier, then the table or timetable uses the modified variable names.

For more information, see `table`, `timetable`, or `Access Data in Tables`.

### **sheetnames Function: Get names of worksheets from spreadsheet file**

Use the `sheetnames` function to get the names of all worksheets from a spreadsheet file. For more information, see `sheetnames`.

### **VideoReader Object: Read frames in videos using frame index or time**

The `VideoReader` object supports interchangeable access to video frames using frame index or time. Therefore, you can now use `read` and `readFrame` interchangeably.

Previously, you could use only one type of access at a time. Attempting to read frames interchangeably using `read` and `readFrame` resulted in an error. For example, the following code would result in an error.

```
vidObj = VideoReader('xylophone.mp4');
frame20 = read(vidObj,20); % read using frame index
frame21 = readFrame(vidObj) % read next frame using CurrentTime
```

Cannot call 'READFRAME' method after querying the NUMBEROFFRAMES property or using the READ method. Recreate the object to use 'READFRAME' method.

However, starting in R2019b, you can use all the methods and properties of the `VideoReader` object interchangeably between accessing frames using frame indices and accessing frames using time. For example, create a `VideoReader` object and read the 20th frame using the `read` method. Next, use the `readFrame` method to get the 21st frame.

```
vidObj = VideoReader('xylophone.mp4');
frame20 = read(vidObj,20);
frame21 = readFrame(vidObj);
whos frame20 frame21
```

Name	Size	Bytes	Class	Attributes
frame20	240x320x3	230400	uint8	
frame21	240x320x3	230400	uint8	

For more information, see `VideoReader`.

## VideoReader Object: Improved performance in generated code with row-major layout

For large video files, the generated code for the `VideoReader` object with a row-major layout option shows improved performance. For example, the `timingTest` function shows about a 4x speed-up on a H.264 video file with a resolution of 1280x720.

```
function [t, data] = timingTest(fileName)
    vidObj = VideoReader(fileName);
    data = cell(20,1);
    tic;
    for cnt = 1:20
        data{cnt} = readFrame(vidObj);
    end
    t = toc;
end
```

Generate code for the `timingTest` function with the row-major flag. The `codegen` command creates a function `timingTest_mex` with the C and C++ generated code.

```
codegen timingTest -args {coder.typeof('', [1 inf])} -rowmajor
```

For a H.264 video file with a resolution of 1280x720, the execution times are:

**R2019a:** 4.04s

**R2019b:** 0.95s

The code was timed on a Windows 10, Intel Xeon® CPU W-2133 @ 3.6 GHz test system by calling the function `timingTest_mex`. The higher the video file resolution (measured by frame size), the greater the performance improvement becomes.

## **Import Tool: Generate simpler code when importing from fixed-width text files**

**Import Tool** now operates consistently across different platforms and generates code that is easy to read when importing fixed-width text files. For example, the generated code contains `readtable` and `FixedWidthImportOptions`, which makes the code simpler to read and use. For more information, see [Read Text File Data Using Import Tool](#).

## **save Function: Save workspace variables to a MAT-file version 7 without compression**

Previously, the `save` command, when saving workspace variables to a version 7 MAT-file, used compression as the default (and the only) option. Now, `save` supports the `'-nocompression'` option for MAT-file version 7.

By default, saving a variable `myVariable` compresses the data and saves it to a version 7 MAT-file. The `-v7` flag is optional.

```
save myFile.mat myVariable -v7
```

To save `myVariable` without compression, use:

```
save myFile.mat myVariable -v7 -nocompression
```

For more information, see `save`.

## **xmlread Function: Prevent reading of XML files that contain DOCTYPE declarations**

You can prevent reading XML files that contain DOCTYPE declarations by setting the `'AllowDoctype'` name-value pair to `false`. For more information, see the `xmlread` reference page.

## **imread Function: Supports reading specified images from PGM, PBM, or PPM file formats**

The `imread` function supports reading specified images from multi-image PGM, PBM, or PPM file formats. For more information, see the `imread` reference page.

## **Scientific File Format Libraries: CFITSIO Library upgraded to version 3.450**

The CFITSIO library is upgraded to version 3.450.



## Scientific File Format Libraries: LibTIFF Library upgraded to version 4.0.10

The LibTIFF library is upgraded to version 4.0.10.

## RESTful Functions: Support for authentication

The RESTful web services functions `webread`, `websave`, and `webwrite` also support Digest authentication. For more information, see the `weboptions` 'Username' argument.

For the list of supported authentications for RESTful functions, see Server Authentication.

## Version History

The RESTful functions `webread`, `webwrite`, and `websave` now adhere more closely to the Internet Engineering Task Force (IETF®) document RFC 7617 for Basic authentication. As a result, MATLAB might error when a RESTful function communicates with a server that proactively expects Basic authentication but does not return a 401 challenge response.

To update your code, see How do I preemptively include a Basic Authentication header when working with "webread"/"webwrite"/"websave" in MATLAB R2019b?

## tcpclient, read, and write Functions: Generate C and C++ code

The `tcpclient`, `read`, and `write` functions support C and C++ code generation using MATLAB Coder.

## Bluetooth Low Energy Interface: Support for scanning and interacting with peripheral devices

You can use MATLAB commands to perform the following operations:

- Scan for nearby peripheral devices and view advertising data using the `blelist` function
- Connect to peripheral devices using the `ble` function
- Access device characteristics and descriptors using the `characteristic` and `descriptor` functions
- Read device characteristic data and descriptor data using the `read` function
- Write to device characteristics and descriptors using the `write` function
- Enable and disable notification or indication for a characteristic using the `subscribe` and `unsubscribe` functions

For more information, see Bluetooth Low Energy Communication.

## Serial Port Devices: New functions and properties

The serial port interface has a new set of functions and properties. The existing functionality still runs, but new function names and properties are recommended. The new interface has improved performance.

Get started with the new interface by viewing a list of all serial ports on your computer using `serialportlist`.

```
list = serialportlist

list =
    1x4 string array
    "COM1"    "COM3"    "COM4"    "COM8"
```

Then, create a `serialport` object, write data to the device, and read from it.

```
s = serialport("COM8",115200);
write(s,1:5,"uint32")
read(s,5,"uint32");
```

For more information, see [Serial Port Devices](#).

## Version History

For more information about updating your code to use the recommended functionality, see [Transition Your Code to serialport Interface](#).

## Functionality being removed or changed

### Import Tool does not support importing text data as cell array of character vectors

*Behavior change*

Previously, **Import Tool** provided an option to import text data as a cell array of character vectors. Starting in R2019b, **Import Tool** does not support this. Instead, the **Import Tool** app imports text data as string arrays.

To preserve the previous behavior, convert the imported text data to a cell array of character vectors using the `cellstr` function.

### UseExcel parameter for spreadsheets

*Behavior change*

The default setting for `UseExcel` on Windows systems with Excel® installed is `false`.

The `UseExcel` parameter appears on these spreadsheet functions: `readtable`, `readtimetable`, `readmatrix`, `readcell`, `readvars`, `writetable`, `writetimetable`, `writematrix`, and `writecell`.

To preserve the previous behavior, update calls to these functions to specify the `UseExcel` parameter as `true`. For example, for the `readtable` function, update your code as follows.

```
T = readtable(filename,'UseExcel',true)
```

### xlsinfo function is not recommended

*Still runs*

The `xlsinfo` function is not recommended. Use the function `sheetnames` instead. There are no plans to remove `xlsinfo` at this time. The `sheetnames` function has these advantages over the `xlsinfo` function:

- Support for sheet names containing non-ASCII characters
- Better cross-platform support and performance
- Ability to work with remotely stored data

This table shows a typical use of `xlsinfo` and how to update your code to use `sheetnames`.

Not Recommended	Recommended
<code>[~,sheets] = xlsinfo('myData.xls')</code>	<code>sheets = sheetnames('myData.xls')</code>

### NumberOfFrames property of the VideoReader object is not recommended

*Still runs*

The `NumberOfFrames` property of the `VideoReader` object is not recommended. Use the property `NumFrames` instead. Update all instances of `NumberOfFrames` with `NumFrames`. There are no plans to remove the `NumberOfFrames` parameter at this time.

### Array of VideoReader objects is not supported

Creating an array of `VideoReader` objects is not supported. Update your code to remove arrays of `VideoReader` objects. For example, the following code returns an error.

```
v = VideoReader('xylophone.mp4');
v(end+1) = VideoReader('xylophone.mpg');
```

Array formation and parentheses-style indexing with objects of class 'VideoReader' is not allowed. Use objects of class 'VideoReader' only as scalars or use a cell array.

### Tiff object for writing certain TIFF files

Writing TIFF images with certain combinations of photometric configuration and the number of samples per pixel is not recommended. The value of `SamplesPerPixel` must be equal to the sum of `Photometric` color channels and the `ExtraSamples` specified in the `Tiff` object. For more information, see `Tiff` and `write`.

### imwrite function does not support writing of indexed PNG files that have insufficient colormap entries

Starting in R2019b, for writing indexed PNG files, you must specify a `colormap` with enough entries to interpret the image correctly. For more information, see the `imwrite` reference page.

### imfinfo function returns information on multiple images from PGM, PBM, and PPM files

*Behavior change*

Previously, for the PGM, PBM, and PPM file formats, the `imfinfo` function returned a single 1-by-1 structure. The structure contained information about only the first image, even if the file contained multiple images in it.

Starting in R2019b, if the PGM, PBM, and PPM files have multiple images, then `imfinfo` returns a structure array containing information on multiple images in the file. For instance, for a PGM file containing `M` images, `imfinfo` returns a 1-by-`M` structure array containing information corresponding to all the images in the file.

This table shows how to update your code to get the `Height` (or any other property) of the first image of a multi-image PBM, PGM, or PPM file.

Not Recommended	Recommended
<code>info = imfinfo('MultiImgFile.pgm')</code> <code>info.Height</code>	<code>info = imfinfo('MultiImgFile.pgm')</code> <code>info(1).Height</code>

For more information, see `imfinfo`.

### web Function

#### *Behavior change*

The `web` function now opens external sites using your system browser by default. Previously, the `web` function opened external sites using the MATLAB browser. Using the system browser is recommended when opening external sites.

To use the MATLAB browser as the default browser for external sites, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web** and in the **System Web browser** section, clear the **Use system web browser when opening links to external sites (recommended)** option.

### Web services use system certificates

#### *Behavior change*

The default value for the `CertificateFilename` option in the `weboptions` function and the `matlab.net.http.HTTPOptions.CertificateFilename` property is `'default'`. If the value is `'default'`, then MATLAB uses system certificates.

Previously by default, MATLAB used the PEM certificate file that ships with MATLAB.

### serialist function is not recommended

#### *Still runs*

`serialist` is not recommended. Use `serialportlist` instead. See [Transition Your Code to serialport Interface](#) for more information about using the recommended functionality.

### serial function is not recommended

#### *Still runs*

`serial` and its object properties are not recommended. Use `serialport` and its properties instead.

This example shows how to connect to a serial port device using the recommended functionality.

Functionality	Use Instead
<code>s = serial("COM1");</code> <code>s.BaudRate = 115200;</code> <code>fopen(s)</code>	<code>s = serialport("COM1",115200);</code>

See [Transition Your Code to serialport Interface](#) for more information about using the recommended functionality.

## Mathematics

### **makima Function: Perform modified Akima cubic Hermite interpolation**

The `makima` function performs modified Akima cubic Hermite interpolation, similar to the 'makima' interpolation method of `griddedInterpolant`, `interp1`, `interp2`, `interp3`, and `interpN`. The modified Akima cubic Hermite interpolation method has these properties:

- It is  $C^1$  continuous.
- It produces fewer undulations than `spline`, but the result is not as aggressively flattened as `pchip`.
- Unlike `pchip`, it supports N-D arrays.
- Unlike `spline`, it does not produce overshoots.

## Graphics

### Chart Container Class: Develop your own class of charts

Define your own class of charts by creating a subclass of the `matlab.graphics.chartcontainer.ChartContainer` base class. If you write scripts or functions for creating specialized visualizations and share them with others, consider creating a class implementation. Creating a class enables you to:

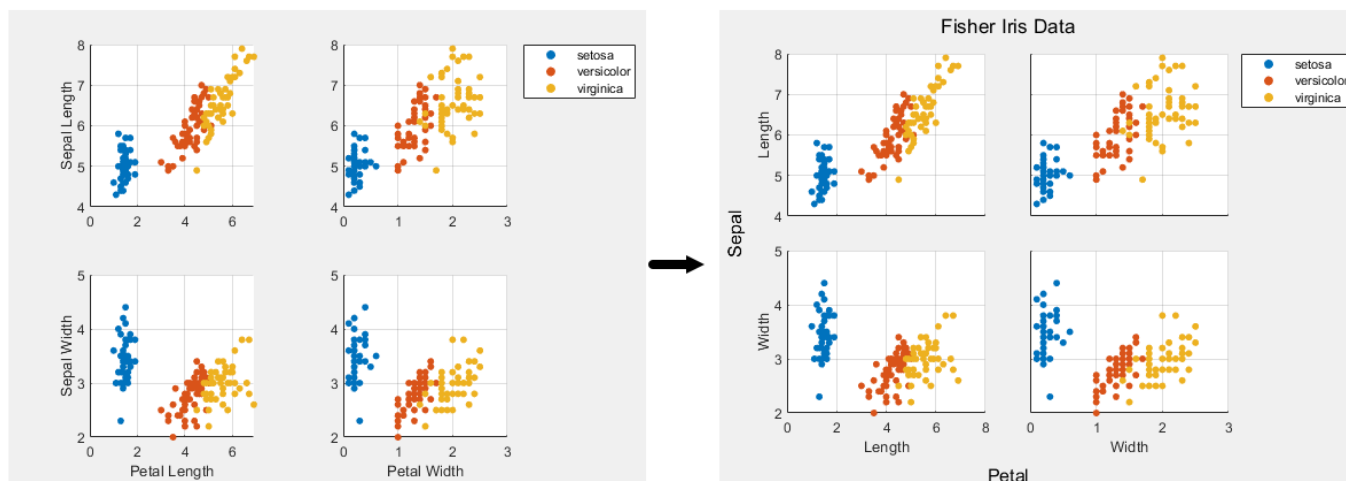
- Provide a convenient interface for your users — When users want to customize an aspect of your chart, they can set a property rather than having to modify and rerun your graphics code. Users can modify properties at the command line or inspect them in the Property Inspector.
- Encapsulate algorithms and primitive graphics objects — You implement methods that perform calculations and manage the underlying graphics objects. Organizing your code in this way allows you to hide implementation details from users.

The `ChartContainer` base class supports charts that have a single Cartesian axes. When you define a chart that derives from this base class, instances of your chart are members of the graphics object hierarchy. As a result, your charts are compatible with many aspects of the graphics system. For example, the `gca` and `findobj` functions can return instances of your chart. For more information, see [Chart Development Overview](#).

### tiledlayout and nexttile Functions: Create configurable layouts of plots in a figure

Use the `tiledlayout` and `nexttile` functions to lay out a tiling of multiple plots within a figure. The configuration options include:

- Control over the spacing between the plots and around the edges of the layout
- An option for a shared title at the top of the layout
- Options for shared x- and y-axis labels
- An option for displaying a shared axes toolbar
- An option to control whether the tiling has a fixed size or variable size that can reflow



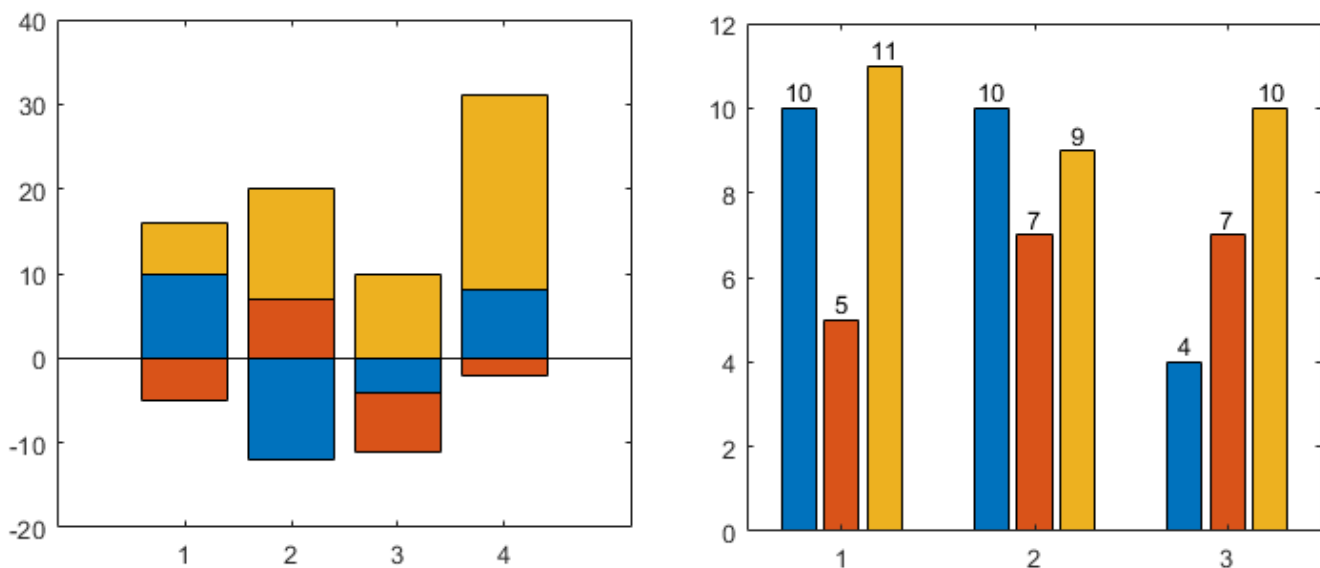
## colororder Function: Control the colors in plots

Control the color scheme of your plots by calling the `colororder` function. When you change the color scheme using this function, the change has an immediate effect on your existing plots. If you call the `colororder` function for an empty figure, the new color scheme persists in the figure when you call a plotting function.

## Bar Charts: Create bar charts with improvements for stacking and locating the tips of bars

The `bar` and `barh` functions have these improvements:

- Stacked groups of bars display negative bars below zero, rather than overlapping the bars.
- You can locate the tips of bars by getting the values of the `XEndPoints` and `YEndPoints` properties on the `Bar` object. The locations are useful for adding labels to the tips of bars.
- Both functions now accept more combinations of scalar, vector, and matrix inputs.



## Data Tips: Create and customize data tips

Create data tips on objects with a `DataTipTemplate` property using the `datatip` function. You can create a data tip by specifying the exact coordinates of the data point, the approximate coordinates of the data point, or the index of the data point within the plotted data set. Create a data tip between plotted data points by setting the `SnapToDataVertex` property to `'off'`.

In the Live Editor, generate code for interactively pinned data tips by clicking the **Update Code** button. This generated code recreates the data tip the next time you run the live script.

Customize the content of data tips on additional charts, including `Contour`, `Patch`, `Quiver`, `Bar`, and `Image` objects with a `DataTipTemplate` property. For more information, see [Create Custom Data Tips](#).







## dataTipInteraction Function: Pin data tips at cursor location

Create data tip interactions that pin data tips to the cursor location, rather than the nearest data point, by setting the `SnapToDataVertex` property of the data tip interaction object to `'off'`:

For more about customizing axes interactions, see [Control Chart Interactivity](#).

## Axes Toolbar: Save or copy contents of axes as image

To save or copy the contents of a set of axes or a tiled chart layout, hover over the **Export** icon  and select an option from the drop-down menu. The available options depend on the content of the axes.

- Save to an image or PDF file by selecting **Save As** .
- Copy to the clipboard as a PNG image by selecting **Copy as Image** .
- Copy to the clipboard as a vector graphic for PDFs by selecting **Copy as Vector** .


When you create a custom toolbar using the `axtoolbar` function, include a drop-down menu with save and copy options by specifying the `buttons` argument as `'export'`.

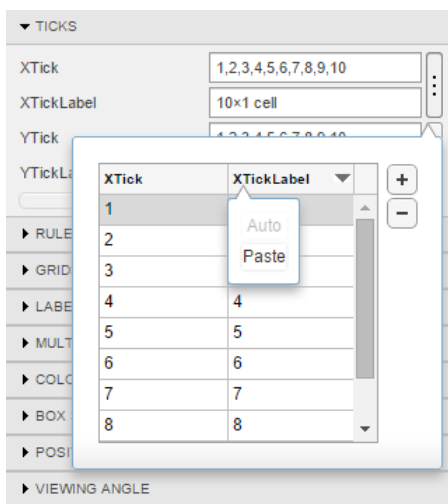
## parallelplot Function: Zoom, pan, and rearrange coordinates interactively

The parallel coordinates plot has new options for interacting with data:

- Zoom — Use the scroll wheel to zoom.
- Pan — Click and drag the plot to pan.
- Rearrange coordinates — Click and drag a coordinate tick label to move the corresponding coordinate ruler to a different position.

## Property Inspector: Update axis tick values and labels using clipboard data

You can update the tick values and labels that appear along an axis by copying and pasting data (for example, from a spreadsheet) into the property inspector. To paste copied data, open the property inspector and navigate to the ticks property that you want to edit. Then, open the drop-down menu by clicking  and select **Paste**.



## Image Interpolation: Select an interpolation method for displaying images

Display images using either nearest neighbor or bilinear interpolation. MATLAB uses interpolation when it displays a scaled or rotated version of an image.

When you call the `image`, `imagesc`, or `imshow` functions, specify the `Interpolation` property name-value pair with either `'nearest'` or `'bilinear'` as the value. All images use `'nearest'` by default.

For example, this code displays an image using the `imagesc` function with bilinear interpolation:

```
I = imread('peppers.png');
imagesc(I, 'Interpolation', 'bilinear')
```

## legend Function: Create unlimited legend entries and specify categorical arrays

The `legend` function has these improvements:

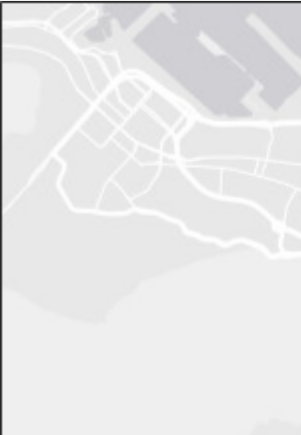

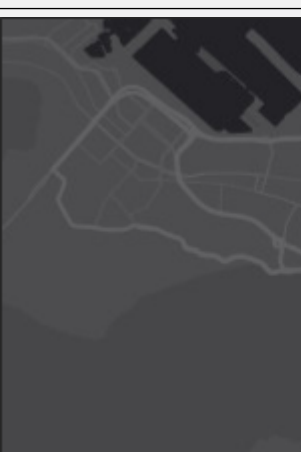

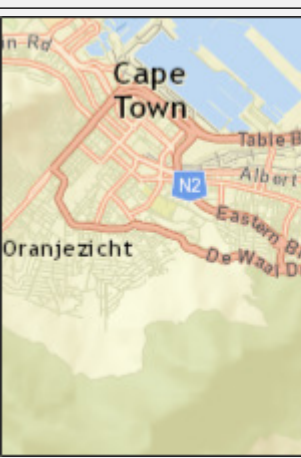
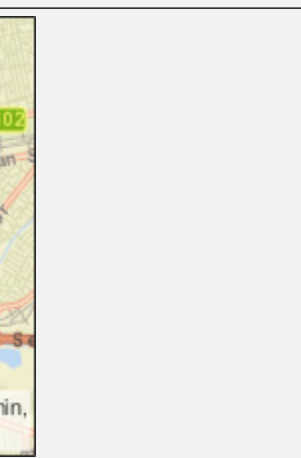

- The legend can display an unlimited number of entries when you specify the `subset` argument. In previous releases, legends display up to 50 entries only.
- You can specify label values as categorical arrays in addition to cell arrays and string arrays.

## pcolor Function: Specify categorical, datetime, and duration data

The `pcolor` function now accepts categorical, `datetime`, and `duration` arrays for `X` and `Y`.

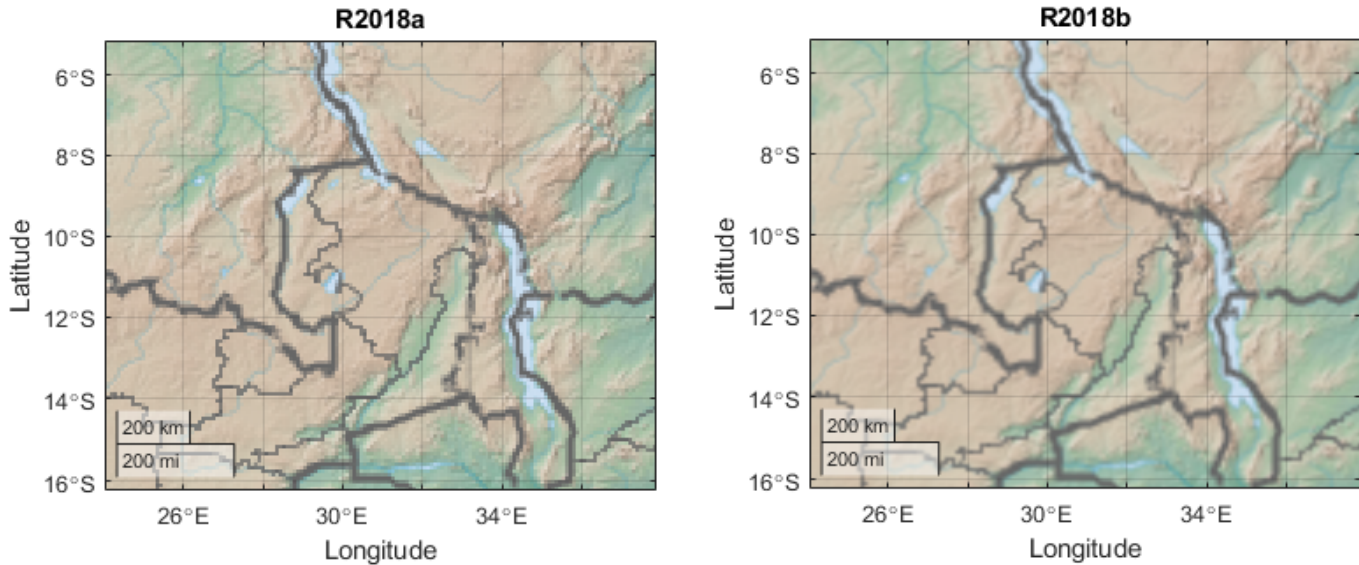
## Geographic Plots: Plot data on high-zoom-level basemaps

Plot data on high-zoom-level basemaps hosted by Esri®. For more information about Esri, see <https://www.esri.com>. Specify a basemap using the `geobasemap` function or by setting the `Basemap` property of the `GeographicAxes` or `GeographicBubbleChart` object.

	<p>'streets-light'</p> <p>Map designed to provide geographic context while highlighting user data on a light background.</p> <p>Hosted by Esri.</p> <p>Esri South Africa, HERE, Garmin, NGA, USGS</p>		<p>'satellite'</p> <p>Full global basemap composed of high-resolution satellite imagery.</p> <p>Hosted by Esri.</p> <p>Earthstar Geographics, CNES/Airbus DS</p>
	<p>'streets-dark'</p> <p>Map designed to provide geographic context while highlighting user data on a dark background.</p> <p>Hosted by Esri.</p> <p>Esri, HERE, Garmin, NGA, USGS</p>		<p>'topographic'</p> <p>General-purpose map with styling to depict topographic features.</p> <p>Hosted by Esri.</p> <p>Esri South Africa, HERE, Garmin, USGS, NGA</p>
	<p>'streets'</p> <p>General-purpose road map that emphasizes accurate, legible styling of roads and transit networks.</p> <p>Hosted by Esri.</p> <p>Esri South Africa, HERE, Garmin, NGA, USGS</p>		

### Geographic Plots: Create plots with improved basemap appearance

Starting in R2018b, basemaps within geographic plots have an improved visual appearance at noninteger zoom levels. For example, the `ZoomLevel` property of these geographic bubble charts is 4.995.



## Geographic Axes: Display animations using comet or animatedline

Display animations on geographic axes using `comet` or `animatedline`. For more information, see [Line Animations](#).

## Geographic Bubble Charts: Create charts with improved layout

Geographic bubble charts have an improved visual appearance, including use of figure space and placement of ticks and tick labels.

When you resize a geographic bubble chart, font sizes and spacing between elements in the chart automatically adjust to provide the best possible presentation for the new size. Changing the `FontSize` property of a geographic bubble chart disables the automatic resizing of the fonts.

## Functionality being removed or changed

### Changing `ColorOrder` or `LineStyleOrder` on the axes affects existing plots immediately

*Behavior change*

If you change the axes `ColorOrder` or `LineStyleOrder` properties after plotting into the axes, the colors and line styles in your plot change immediately. In previous releases, the new colors and line styles affect only subsequent plots, not the existing plots.

For example, this code plots two lines before changing the value of the `ColorOrder` property. In R2019a and previous releases, the colors of the two lines do not change when you change the `ColorOrder` property.

```
plot([0 1],[1 0])
hold on
plot([0 1],[2 0])
ax = gca;
ax.ColorOrder = [1 0 0; 0 1 0];
```

Starting in R2019b, the lines change immediately to use the new `ColorOrder` or `LineStyleOrder` property values. If you want to prevent the change, set either the axes `LineStyleOrderIndex` or `ColorOrderIndex` property to any value (such as its current value) before changing the `ColorOrder` or `LineStyleOrder` property.

```
plot([0 1],[1 0])
hold on
plot([0 1],[2 0])
ax = gca;

% Preserve R2019a behavior
ax.ColorOrderIndex = ax.ColorOrderIndex;

ax.ColorOrder = [1 0 0; 0 1 0];
```

### Indexing scheme for `ColorOrder` and `LineStyleOrder` might change plot colors and line styles

#### *Behavior change*

In R2019a and previous releases, plots that visualize multiple sets of data rely on an indexing scheme to select the colors and line styles. The indexing scheme generally works well, but it does not allow you to change the colors and line styles in a plot after you create it.

Starting in R2019b, there is a new indexing scheme that enables you to change the colors and line styles of existing plots. MATLAB always applies this indexing scheme to objects such as `Line`, `Scatter`, and `Bar`. As a result, your code might produce plots that cycle through the colors and line styles differently than in previous releases.

One example of the new behavior is when you create a line with a specific color, and then create another line without specifying a color. In the following code, the first call to the `plot` function includes a third argument that specifies the color as red. The second call to the `plot` function does not specify a color. In R2019a, the first line is red, and the second line is blue.

```
% Plot two lines
hold on
plot([0 1],[1 0],'red')
plot([0 1],[2 0])
```

If you run the preceding code in R2019b, the first line is red and the second line is orange. To preserve the original color, set either the axes `LineStyleOrderIndex` or `ColorOrderIndex` property to any value (such as its current value) before plotting into the axes.

```
% Preserve R2019a behavior
ax = gca;
ax.ColorOrderIndex = ax.ColorOrderIndex;

% Plot two lines
hold on
plot([0 1],[1 0],'red')
plot([0 1],[2 0])
```

### Predefined colormaps have 256 colors by default

#### *Behavior change*

The predefined colormaps, such as `parula`, `jet`, and `winter`, now have 256 colors by default.

If you have code that depends on a predefined colormap having 64 colors, specify the number of colors when you set the colormap for the figure, axes, or chart. For example, `colormap(parula(64))` sets the figure's colormap to the 64-color `parula` colormap.

Alternatively, you can change the default colormap for all figures within your MATLAB session:

```
set(groot, 'defaultFigureColormap', parula(64))
```

### **Pie charts display zero values**

#### *Behavior change*

When you call the `pie` or `pie3` function and specify data that contains zero values, your pie chart shows the zero values and corresponding labels. If you call either function with an output argument, the output includes objects for each zero value.

In previous releases, the functions omit the zero values from the chart and do not return any objects that correspond to those values. If you do not want to display zero values or return the corresponding objects, then remove the zeros from your data.

### **Using the axis function to set axes limits no longer changes the view of the plot box**

#### *Behavior change*

When you call the `axis` function to set limits, the plot box no longer changes its view in these situations:

- When the plot box is in a 2-D view, and you pass a six-element vector to the `axis` function. To preserve the behavior of previous releases, call `view(3)` after calling the `axis` function.
- When the plot box is in a 3-D view, and you pass a four-element vector to the `axis` function. To preserve the behavior of previous releases, call `view(2)` after calling the `axis` function.

### **Default basemap for geographic plots is now 'streets-light'**

#### *Behavior change*

Starting in R2019b, the default basemap for `GeographicAxes` and `GeographicBubbleChart` objects is `'streets-light'`. The `'streets-light'` basemap requires Internet access.

In previous releases, the default basemap was `'darkwater'`. This basemap is included with MATLAB and does not require Internet access.

If you do not have reliable access to the Internet, you can use the `'darkwater'` basemap or download a selection of basemaps onto your local system using the Add-On Explorer. For more information, see `Access Basemaps in MATLAB`.

### **Turning on interaction modes disables mouse interactions for geographic bubble charts**

#### *Behavior change*

Starting in R2019b, turning on an interaction mode in a figure disables mouse interactions for geographic bubble charts in the figure. For example, if you turn on zoom mode using the `zoom` function, then you can no longer use the mouse to zoom or pan within the geographic bubble chart. Other interaction modes include pan, rotate, data cursor, or brush mode.

To zoom or pan within a geographic bubble chart, turn off interaction modes within the figure and zoom or pan using the built-in mouse interactions.

**Geographic bubble charts display tick labels using seconds***Behavior change*

Starting in R2019b, the tick label format for geographic bubble charts is degrees, minutes, and decimal seconds (DMS) rather than degrees and decimal minutes (DM).

## App Building

### **uistyle Function: Create styles for rows, columns, or cells in a table UI component**

You can create different styles for specific rows, columns, or cells in a table UI component using the `uistyle` and `addStyle` functions. For example, you can make the cells in a specific column red with italic font. To retrieve styles that have been applied, get the `StyleConfigurations` property of the `Table` object. To remove a style from a table UI component, use the `removeStyle` function.

Cell styles in table UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

### **uigridlayout Function: Configure grid rows and columns to adjust automatically to fit components**

Grid layouts can automatically adjust to the minimum size needed to fit the components that are in it. To configure grid rows and columns to fit components dynamically, specify `'fit'` as the value of the `RowHeight` or `ColumnWidth` properties for specific rows and columns in the `GridLayout` object. For example, setting `'RowHeight'` to `{'fit', 50, '1x'}` specifies that the height of the first row is tightly fit around the components, the second row is fixed at 50 pixels, and the third row uses the remaining vertical space.

This feature is especially useful when you are creating rows and columns of text-based components because when you use `'fit'`, you don't have to know how tall or wide to make the rows and columns of the grid layout manager. The `'fit'` option for row height and column width is also useful if your app is translated to another language or runs on different platforms.

The `uigridlayout` function is supported only in App Designer apps and in figures created with the `uifigure` function.

### **uitable Function: Sort table UI components interactively when using logical, numeric, string, or cell arrays**

You can interactively sort table UI components when the `Data` property contains logical data, numeric data, string data, or cell array data. To sort table UI components that contain these data types, set the `ColumnSortable` property of the `Table` object. To update visualizations based on how a table UI component was sorted, also use the `DisplayData` property and a `DisplayDataChangedFcn` callback function.

Sortable columns in table UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

### **uihtml Function: Embed HTML, JavaScript, or CSS content in apps and on the App Designer canvas**

To embed HTML, JavaScript®, or CSS in your app, call the `uihtml` function or, in App Designer, drag an HTML UI component from the **Component Library** onto the canvas.




HTML UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

## App Designer: Convert components in a UI figure or container from pixel-based positioning to a grid layout manager

You can convert the children of a UI figure or container from pixel-based positioning to being positioned by a grid layout manager. To use a grid layout manager where you were previously using pixel-based positioning, drag a grid layout from the **Component Library** onto the canvas or into an existing container component, like a panel. Alternatively, right-click the canvas or container component and select **Apply Grid Layout**.

The grid layout manager automatically creates rows and columns to accommodate the components, and preserves their approximate positions. When you add a grid layout, the component hierarchy updates in the **Component Browser**.

## App Designer: Convert an existing app into an auto-reflowing app

To convert an existing app into an auto-reflowing app, expand the **Convert**  drop-down menu in the **Canvas** tab, and select **2-Panel App with Auto-Reflow** or **3-Panel App with Auto-Reflow**.

Auto-reflowing apps automatically resize and reflow content based on screen size, screen orientation, and platform. Use apps with auto-reflow if you expect to run or share your apps across multiple environments or desktop resolutions. For more details, see [Apps with Auto-Reflow](#).

## App Designer: Suppress Code Analyzer warning messages

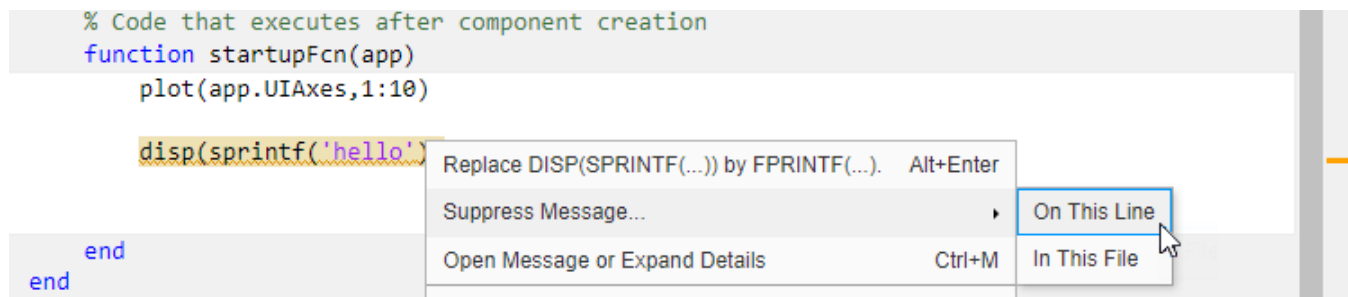
In the App Designer **Code View** editor, you can suppress Code Analyzer warnings for a single line or for the entire app file. To suppress warnings, right-click a warning and, from the context menu, select:

- **Suppress Message... > On This Line**
- **Suppress Message... > In This File**

For example,


```
% Code that executes after component creation
function startupFcn(app)
    plot(app.UIAxes,1:10)

    disp(sprintf('hello'))
end
end
```



Error messages, however, cannot be suppressed.

## App Designer: Open App Designer from the MATLAB toolstrip

To open App Designer from the MATLAB toolstrip, click the **Design App**  button in the **Apps** tab.

## App Testing Framework: Perform gestures on polar axes and UI images

The App testing framework supports gestures on more UI components.

- Perform hover and press gestures in tests on polar axes.
- Perform press gestures in tests on UI images.

For example, perform interactive gestures on PolarAxes object `pax` and Image object `im`.

```
fig = uifigure;
pax = polaraxes(fig, 'ThetaAxisUnits', 'radians');
im = uiimage(fig, 'ImageSource', 'membrane.png', 'Position', [10 10 100 100]);
testCase = matlab.uitest.TestCase.forInteractiveUse;
testCase.hover(pax);
testCase.press(pax, [pi/2 0.5]);
testCase.press(im);
```

For more information, see the hover and press reference pages.

## Functionality being removed or changed

### GUIDE will be removed in a future release

*Still runs*

The GUIDE environment and the `guide` function will be removed in a future release.

After GUIDE is removed, existing GUIDE apps will continue to run in MATLAB but will not be editable using the drag-and-drop environment in GUIDE. To continue editing an existing GUIDE app and help maintain its compatibility with future MATLAB releases, use one of the suggested migration strategies listed in the table.

App Development	Migration Strategy	How to Migrate
Frequent or ongoing development	Migrate your app to App Designer	Use the GUIDE to App Designer Migration Tool for MATLAB on <a href="https://www.mathworks.com">mathworks.com</a>
Minimal or occasional editing	Export your app to a single MATLAB file to manage your app layout and code using MATLAB functions	Open the app in GUIDE and select <b>File &gt; Export to MATLAB-file</b>

App Designer is the recommended app development environment in MATLAB. To create new apps, use App Designer and the `appdesigner` function instead.

To learn more about migrating apps, see [GUIDE Migration Strategies](#).

For more information about App Designer, go to [Comparing GUIDE and App Designer on mathworks.com](#).

**javacomponent function and JavaFrame property will be removed in a future release***Warns*

The `javacomponent` function and the `JavaFrame` figure property are undocumented and will be removed in a future release. Update your code to use documented alternatives. For a list of documented functionality that you can use instead, see Recommendations for Java and ActiveX Users on [mathworks.com](https://www.mathworks.com).

**actxcontrol, actxcontrollist, and actxcontrolselect functions will be removed in a future release***Warns*

The `actxcontrol`, `actxcontrollist`, and `actxcontrolselect` functions will be removed in a future release. Update your code to use alternate functionality. For a list of functionality that you can use instead, see Recommendations for Java and ActiveX Users on [mathworks.com](https://www.mathworks.com).

**Support for running deployed web apps in Internet Explorer has been removed***Errors*

Support for running deployed web apps in Internet Explorer® has been removed. Use the current versions of Google Chrome (recommended), Safari, Firefox, or Microsoft Edge to run deployed web apps instead.

For more information on supported web app browsers, see Supported Browsers and Platform Incompatibilities (MATLAB Compiler).

**Text alignment and font size have changed in table column and row headers***Behavior change*

Starting in R2019b, table UI components created in App Designer or in figures created with the `uifigure` function have a different visual appearance when they contain certain kinds of data. Column and row headers of table UI components that contain numeric, logical, string, or cell array data have these visual differences compared to previous releases:

- Smaller font size
- Column headers are left-aligned instead of center-aligned
- Row headers are center-aligned instead of left-aligned

For example, this code that creates a table UI component with mixed cell array data renders differently in R2019b than it does in R2019a.

```
fig = uifigure;

d = {'Male',52,true;'Male',40,true;'Female',25,false};

uit = uitable(fig,'Data',d);
uit.ColumnName = {'Gender','Age','Authorized'};
```

## R2019b:

	Gender	Age	Authorized
1	Male	52	<input checked="" type="checkbox"/>
2	Male	40	<input checked="" type="checkbox"/>
3	Female	25	<input type="checkbox"/>

## R2019a:

	Gender	Age	Authorized
1	Male	52	<input checked="" type="checkbox"/>
2	Male	40	<input checked="" type="checkbox"/>
3	Female	25	<input type="checkbox"/>

Starting in R2019a, the same visual differences apply to column and row headers in table UI components that contain table array data. For example, this code that uses a table array to display `datetime` values in a table UI component renders differently in R2019a than it does in R2018b.

```
fig = uifigure;

dates = datetime([2016,01,17; 2017,01,20], 'Format', 'MM/dd/yyyy');
m = [10; 9];
tdata = table(dates,m, 'VariableNames', {'Date', 'Measurement'});

uit = uitable(fig, 'Data', tdata);
uit.RowName = 'numbered';
```

## R2019a:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

## R2018b:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

## Performance

### table Data Type Indexing: Improved performance when assigning elements by subscripting into large table variables

table subscripted assignment into large table variables is significantly faster. Performance is now essentially constant with the number of elements in each table variable.

- For example, when you use dot indexing to assign elements to a variable with  $10^6$  elements, performance in R2019b is approximately 40x times faster, as shown below.

```
function timingTest()
    t = table(zeros(1e6,1));
    indices = randi(1e6,1,10000);

    tic;
    for i = indices
        t.Var1(i) = rand;
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 47.83 s

**R2019b:** 1.20 s

- Similarly, assignment using curly braces is faster. For example, when you assign into three table variables with  $10^6$  elements, performance in R2019b is approximately 18x faster.

```
function timingTest()
    t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
    indices = randi(1e6,1,10000);

    tic;
    for i = indices
        t{i,:} = rand;
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 156.39 s

**R2019b:** 8.51 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the `timingTest` function.

The larger the table variables are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make table subscripted assignments within a function. There is no improvement when subscripting into tables at the command line, or within try-catch blocks.

## datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting into large arrays

datetime, duration, and calendarDuration subscripted assignment into large arrays is significantly faster. Performance is now essentially constant with the number of elements in an array.

- For example, when you assign into a datetime array with  $10^6$  elements, performance in R2019b is approximately 106x times faster, as shown below.

```
function timingTest()
    dt = datetime + hours(1:1e6);
    indices = randi(1e6,1,10000);
    rhs = NaT;

    tic;
    for i = indices
        dt(i) = rhs;
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 49.00 s

**R2019b:** 0.46 s

- Similarly, assignment into a duration array is faster. For example, when you assign into a duration array with  $10^6$  elements, performance in R2019b is approximately 106x times faster.

```
function timingTest()
    d = hours(1:1e6);
    indices = randi(1e6,1,10000);

    tic;
    for i = indices
        d(i) = NaN;
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 48.66 s

**R2019b:** 0.46 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the timingTest function.

The larger the arrays are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make subscripted assignments within a function. There is no improvement when subscripting into datetime, duration, and calendarDuration arrays at the command line, or within try-catch blocks.

## datetime Data Type Indexing: Improved performance when referring or assigning to date and time components of datetime arrays

Subscripted references and assignments to components of `datetime` arrays is significantly faster.

- For example, when you refer to a component of a `datetime` array with  $10^4$  elements, performance in R2019b is approximately 25x times faster, as shown below.

```
function timingTest()
    dt = datetime + hours(1:1e4);
    indices = randi(1e4,1,10000);

    tic;
    for i = indices
        x = dt.Hour(i);
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 12.97 s

**R2019b:** 0.52 s

- Similarly, assignment into a component of a `datetime` array is faster. For example, when you assign into a component of a `datetime` array with  $10^4$  elements, performance in R2019b is approximately 32x times faster.

```
function timingTest()
    dt = datetime + days(1:1e4);
    indices = randi(1e4,1,10000);

    tic;
    for i = indices
        dt.Hour(i) = 0;
    end
    toc
end
```

The approximate execution times are:

**R2019a:** 22.51 s

**R2019b:** 0.70 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the `timingTest` function.

The larger the arrays are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make subscripted assignments within a function. There is no improvement when subscripting into `datetime` arrays at the command line, or within try-catch blocks.

## uitable Function: Faster performance when data type is numeric, logical, or a cell array of character vectors

Tables created with the `uitable` function have better rendering performance, and higher frame rates while scrolling when they contain certain kinds of data. The improvements occur when the `Data` property contains numeric data, logical data, or a cell array of character vectors. The table must be parented to a figure created with the `uifigure` function, or one of its child containers.

Tables containing these data types render up to 40% faster, and interaction performance (like scrolling) is up to 75% faster. For example, on a test system, this code that uses numeric data renders the table faster in R2019b than in previous releases.

```
rows = 10000;  
columns = 25;  
ndata = randi(30,[rows columns]);  
fig = uifigure;  
uit = uitable(fig, 'Data', ndata);
```

## unzip and gunzip Functions: Improved performance when extracting contents of zip files and GNU zip files

Extracting the contents of zip files and GNU zip files using `unzip` and `gunzip` is significantly faster when extracting files on network drives.

- For example, when you extract the contents of the example zip file `myarchive.zip` with a file size of 53 MB on a network drive, performance in R2019b is approximately 1.5x faster, as shown below.

```
function timingTest()  
    unzip myarchive.zip;  
end
```

The approximate execution times are:

**R2019a:** 3.06 s

**R2019b:** 2.03 s

- Similarly, when you extract the contents of the example GNU zip file `myotherarchive.gz` with a file size of 27 MB on a network drive, performance in R2019b is approximately 2x faster, as shown below.

```
function timingTest()  
    gunzip myotherarchive.gz;  
end
```

The approximate execution times are:

**R2019a:** 37.22 s

**R2019b:** 18.22 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU across a Gigabit Ethernet connection using the `timeit` function:

```
timeit(@timingTest)
```



Results vary depending on several factors, including connection speed and whether the network files are cached on the system.

## Software Development Tools

### Unit Testing Framework: Run tests in parallel with your custom plugins

You can now run unit tests in parallel when you extend the `TestRunner` instance with your custom plugins. For more information, see [Run Tests in Parallel with Custom Plugin](#).

### Unit Testing Framework: Validate count in string constraints

The `ContainsSubstring`, `IsSubstringOf`, and `Matches` constraints can now count a string scalar or character vector for a given number of times. To specify the number of occurrences, use the `'WithCount'` parameter. For example:

```
import matlab.unittest.constraints.*
testCase = matlab.unittest.TestCase.forInteractiveUse;
testCase.verifyThat('This is long',ContainsSubstring('is','WithCount',2))
```

```
Verification passed.
```

```
testCase.verifyThat('Gain Main Rain',Matches('[GMR]ain','WithCount',2))
```

```
Verification failed.
```

```
-----
Framework Diagnostic:
-----
Matches failed.
--> Actual count does not match expected count.

    Actual Count:
         3
    Expected Count:
         2

Actual char:
    Gain Main Rain
Regular Expression:
    [GMR]ain
```

### Performance Testing Framework: Visually compare two TimeResult arrays

The `matlab.perftest.TimeResult` class has a new method `comparisonPlot`, which enables you to visually compare the time measurement results of two equal-sized sets of performance tests.

### App Testing Framework: Perform gestures on polar axes and images

The app testing framework supports gestures on more UI components.

- Perform hover and press gestures in tests on polar axes.
- Perform press gestures in tests on images.

## Projects: Delete project definition files

You can now use `matlab.project.deleteProject` to easily stop managing your folder with a project and delete all related project definition files without affecting the remaining files.

## Compare Git Branches: Show differences and save copies

In a project under Git source control, you can now select any two revisions and examine file differences. You can show differences between two development branches and save a copy of the selected file on either branch. See [Compare Branches](#).

## Functionality being removed or changed

### Character vectors are no longer equivalent to enumerations in qualifications

*Behavior change*

Starting in R2019b, actual and expected values in qualifications must have the same type when the expected value is an enumeration of a handle class. For example, consider this enumeration class:

```
classdef MyClass < handle
    enumeration
        X
        Y
    end
end
```

The following test fails because 'X' does not represent the enumeration `MyClass.X`:

```
testCase = matlab.unittest.TestCase.forInteractiveUse;
testCase.verifySameHandle('X',MyClass.X)
```

Verification failed.

```
-----
Framework Diagnostic:
-----
verifySameHandle failed.
--> Values do not refer to the same handle.
--> Value must be a handle object. It is of class "char".
--> Classes do not match.
    Actual Value class      : [char]
    Expected Handle Object class : [MyClass]

Actual char:
    X
Expected Handle Object:
    MyClass enumeration

    X
```

In previous releases, the test passed because MATLAB treated 'X' as a representation of the expected enumeration. This change of behavior affects tests using the `IsSameHandleAs` constraint class or the following qualification methods: `verifySameHandle`, `assumeSameHandle`, `assertSameHandle`, `fatalAssertSameHandle`, `verifyNotSameHandle`, `assumeNotSameHandle`, `assertNotSameHandle`, and `fatalAssertNotSameHandle`.

## External Language Interfaces

### **C++ Interface: Options for publishing C++ interface library**

MATLAB automatically renames classes, functions, enums, and member functions with C++ names that are invalid in MATLAB using the `matlab.lang.makeValidName` function. For example, MATLAB converts the class name `_myclass` in library `mylib` to `x_myclass`. As of R2019b, you can modify `x_myclass` in the library definition file. For example, you can change the name to `myclass`. When you use the class in MATLAB, type `clib.mylib.myclass`. Renaming C++ namespaces or the MATLAB package is not supported.

To specify the shape for object pointer types as scalar for all functions in a library, use the name-value pair argument `'TreatObjectPointerAsScalar'` when building the library. To specify the shape for `const char *` pointer types as scalar for all functions, use the `'TreatConstCharPointerAsString'` argument.

To provide a list of macro definitions, use the name-value pair argument `DefinedMacros` when building the library. To provide a list of macro cancellations, use the `UndefinedMacros` argument.

For more information, see `clibgen.generateLibraryDefinition` and `clibgen.buildInterface`.

### **C++ Interface: nullptr supported as output argument**

As of R2019b, the “C++ interface returns type-specific empty values for `nullptr`” on page 6-50. To test for fundamental `nullptr` types, call the `isempty` function. To test for `nullptr` objects, call the `clibIsNull` function.

### **C++ Interface: Read-only (const) object support**

As of R2019b, the “C++ interface treats read-only objects like C++” on page 6-50. To determine if a C++ object is read-only, call the `clibIsReadOnly` function.

### **Java Interface: JRE version 1.8.0\_202 support**

The MATLAB interface to Java supports JRE version 1.8.0\_202, providing improved security and access to new Java features.

### **Out-of-Process Execution of C++ MEX Functions: Customize environment variables**

To customize the environment of a MEX host process that you use to execute a MEX function, call `mexhost` with the `"EnvironmentVariables"` argument.

### **HTTP Web Services: Server authentication support for NTLM and Kerberos protocols**

The HTTP interface also supports these protocols for server authentication.

- Windows — NTLM and Kerberos
- Linux and macOS — NTLM

For more information, see Server Authentication.

## HTTP Web Services: Timeout options

MATLAB has new timeout options for transmitting messages using the HTTP interface.

- `DataTimeout` — timeout in seconds between packets on the network
- `KeepAliveTimeout` — how long the connection to the server stays open after an initial connect, enabling multiple successive messages to be sent over the same connection
- `ResponseTimeout` — seconds to wait for the header of the response from the server after sending the last packet of a request

For more information, see `matlab.net.http.HTTPOptions`.

## Python Interface: Execute Python functions out of process

Run Python functions in processes that are separate from the MATLAB process. For more information, see Out-of-Process Execution of Python Functionality. Use this mode to call functions in third-party libraries that are not compatible with MATLAB.

## Python Interface and Engine: Version 3.5 support discontinued

Support for Python version 3.5 is discontinued.

## Version History

To ensure continued support for your applications, upgrade to a supported version of Python, version 3.6 or 3.7.

## Perl 5.30.1: MATLAB support on Windows

As of R2019b Update 3, MATLAB on Windows ships with Perl version 5.30.1.

- See [www.perl.org](http://www.perl.org) for a standard distribution of perl, perl source, and information about using perl.
- See <https://metacpan.org/pod/HTML::Parser> for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See <https://metacpan.org/pod/HTML::Tagset> for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

## Version History

If you use the `perl` command on Windows platforms, see [www.perl.org](http://www.perl.org) for information about using this version of the Perl programming language.

## Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Microsoft Visual Studio 2019 for C and C++	Windows
Discontinued	Intel Parallel Studio XE 2015 and XE 2016 for Fortran	Windows macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see [Supported and Compatible Compilers](#).

## Functionality being removed or changed

### C++ interface treats read-only objects like C++

*Behavior change*

A C++ read-only object is an object declared with the C++ `const` attribute. You might get such an object as the output of a function or as a data member of a class. Starting in R2019b, the C++ interface for MATLAB honors the `const` type qualifier ensuring that the MATLAB behavior matches the C++ behavior of `const`-qualified types. MATLAB throws an error if you use a read-only object as follows:

- Passing the object to functions with non-`const` inputs
- Calling a non-`const` method on the object
- Modifying object properties

To test if an object is read-only, call the `clibIsReadOnly` function.

In R2019a, the interface ignores the `const` type qualifier, allowing the MATLAB user to utilize `const` objects incorrectly.

### C++ interface returns type-specific empty values for `nullptr`

*Behavior change*

Starting in R2019b, the C++ interface returns type-specific empty values for functions that return `nullptr`. For more information about `nullptr` return types, see [MATLAB to C++ Data Type Mapping](#).

- For type `double`, MATLAB continues to return `[]` for the value `double.empty`.
- For all other fundamental types, MATLAB returns an `MATYPE.empty` value. To test for `nullptr` types, call the `isempty` function.
- For nonfundamental types, MATLAB returns a `nullptr` object. To test for `nullptr` objects, call the `clibIsNull` function.

In R2019a, for fundamental and nonfundamental types, the interface returns a `double.empty([])` value.

For example, suppose that these C++ functions return `nullptr`:

```
class A {
public:
```

```

    double val;
};

// Function returning nullptr object
A* returnNullptrObject() {
    return nullptr;
}

// Functions returning nullptr primitive type ptr
double* returnDoubleNullptr () {
    return nullptr;
}

const char* returnStringNullptr () {
    return nullptr;
}

```

R2019a	R2019b
For objects, MATLAB returns []. (double.empty). nullReturn = clib.nullptr.returnNullptrObject nullReturn = []	MATLAB returns nullptr for an object of class A. nullReturn = clib.nullptr.returnNullptrObject nullReturn = null A
For fundamental types, MATLAB returns []. nullReturn = clib.nullptr.returnStringNullptr nullReturn = []	MATLAB returns empty string array for type const char*. nullReturn = clib.nullptr.returnStringNullptr nullReturn = 0x0 empty string array
For type double, MATLAB returns []. nullReturn = clib.nullptr.returnDoubleNullptr nullReturn = []	No change. MATLAB returns []. nullReturn = clib.nullptr.returnDoubleNullptr nullReturn = []

### pyversion is not recommended

*Still runs*

pyversion is not recommended. Use pyenv instead. There are no plans to remove pyversion at this time.

To execute Python functions out of process, MATLAB provides a new function, pyenv. This function configures Python environment settings, including the version. Even if you do not use the out-of-process feature, MathWorks recommends using pyenv for managing Python settings. For more information, see Out-of-Process Execution of Python Functionality.

### C MEX and engine applications: true, false, and bool defined by <stdbool.h>

*Behavior change*

The definition for true, false, and bool has changed for building MEX files and standalone MATLAB engine and MAT-file applications with C99 compatible compilers on Windows and Linux platforms. MATLAB defines these values using <stdbool.h> as defined by IEEE Std 1003.1:

The `<stdbool.h>` header shall define the following macros:

```
bool
    Expands to _Bool.
true
    Expands to the integer constant 1.
false
    Expands to the integer constant 0.
_bool_true_false_are_defined
    Expands to the integer constant 1.
```

In R2019a and earlier, MATLAB defined these values on Windows and Linux platforms as:

- `true` — #defined as `1`
- `false` — #defined as `0`
- `bool` — typedef as `unsigned char`

For Apple macOS platforms, there is no change.

**actxcontrol, actxcontrollist, and actxcontrolselect functions will be removed in a future release**

*Warns*

The `actxcontrol`, `actxcontrollist`, and `actxcontrolselect` functions will be removed in a future release. MATLAB will support COM server objects only.



# R2019a

---

**Version: 9.6**

**New Features**


**Bug Fixes**

**Version History**

## Environment

### Live Editor Controls: Add check boxes, edit fields, and buttons to set variable values and run the live script

You can add check boxes and edit fields to your live scripts to interactively set variable values. You also can add a button to run the live script when clicked.

To add a check box, edit field, or button, go to the **Live Editor** tab, click  **Control** ▾, and select from the available controls. For more information, see [Add Interactive Controls to a Live Script](#).

### Live Editor Controls: Specify what code to run when a control value changes




By default, when you change the value of an interactive control, the Live Editor runs the section that contains the control. You can now configure an interactive control to run all sections, run the current section and all remaining sections, or to do nothing.

To configure the control, right-click the control and select **Configure Control**. Then, in the **Execution** section, select from the available options.

Configuring an interactive control to do nothing when changed is useful when your live script contains multiple interactive controls and you only want to run the code after changing all of their values. Add a button to the live script to run the code when clicked.

### Live Editor Controls: Hide code when sharing and exporting live scripts with interactive controls

You can hide the code in a live script, showing only the interactive controls, output, and formatted text. Hiding the code is useful when sharing and exporting live scripts.

To hide the code in a live script, click the hide code  button to the right of the live script. To show the code again, click the output inline  button or the output on right  button.

If you export the live script to PDF, HTML, LaTeX, or Microsoft Word, the code remains hidden.

### Live Editor Export: Save live scripts and functions as Microsoft Word documents

To create editable, static documents capable of being viewed outside of MATLAB, save live scripts and functions as Microsoft Word documents. To save a live script or function as a Microsoft Word document, on the **Live Editor** tab, select **Save > Export to Word**. This format is only available on Windows platforms.

For more information about sharing live scripts and functions, see [Share Live Scripts and Functions](#).

## Live Editor Output: Enable animations in plots to show changes in data over time

You can enable for-loop animations in the Live Editor to show changes in plotted data over time.

To enable animations in the Live Editor, set the `matlab.editor.AllowFigureAnimations` setting to `true`:

```
s = settings;  
s.matlab.editor.AllowFigureAnimation.PersonalValue = true;
```

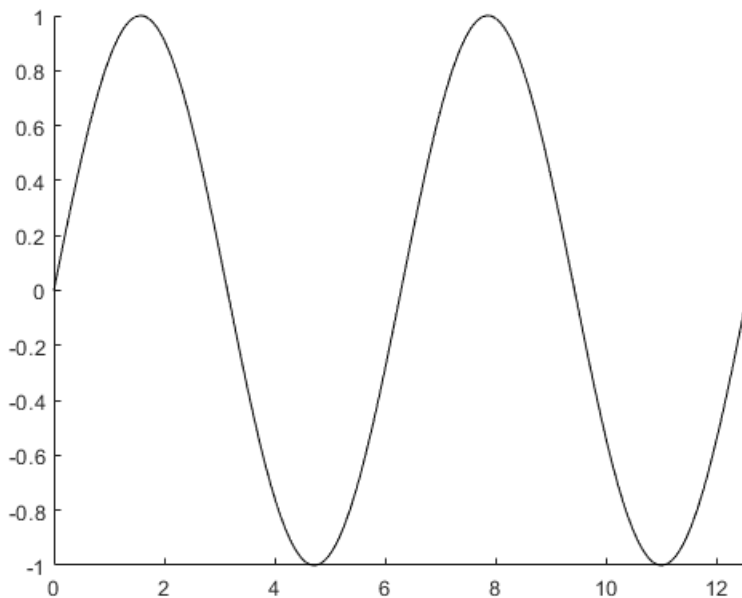
---

**Note** Enabling animations disables support for uicontrols in the Live Editor.

---

For example, this code turns on animations in the Live Editor, and then animates a line growing as it accumulates 2,000 data points. The `drawnow` function displays the changes after each iteration through the loop.

```
s = settings;  
s.matlab.editor.AllowFigureAnimation.PersonalValue = true;  
  
h = animatedline;  
axis([0 4*pi -1 1])  
x = linspace(0,4*pi,2000);  
  
for k = 1:length(x)  
    y = sin(x(k));  
    addpoints(h,x(k),y);  
    drawnow  
end
```



## Live Editor Output: Interactively clean categorical data and filter datetime and duration variables in table output

In the Live Editor, you can interactively clean categorical data and filter datetime and duration variables in table output.


To clean a categorical variable in a table, click the down arrow ▼ to the right of the variable name and select **Edit Categories**. Use the available options to create, remove, and merge categories.

T = 5x3 table

	Temps	Dates	Stations ▼
1	58	'2017-04-17'	S1
2	72	'2017-04-18'	S2
3	56	'2017-04-30'	S1
4	90	'2017-05-01'	S3
5	76	'2017-04-27'	S2

↑ Sort A to Z

↓ Sort Z to A

 Edit Categories

Search

Select All Clear All

<input checked="" type="checkbox"/>	<undefined>	0
<input checked="" type="checkbox"/>	S1	2
<input checked="" type="checkbox"/>	S2	2
<input checked="" type="checkbox"/>	S3	1

Selecting 5 out of 5 rows  
categorical

To filter a datetime or duration variable in a table, click the down arrow ▼ to the right of the variable name and select from the available filtering options.

To add the generated code to your live script, click the **Update Code** button below the table. Adding the generated code to your live script ensures that the cleaning and filtering is reproduced the next time you run the live script.

## Live Editor Output: Interactively change the data type of variables in table output

In the Live Editor, you can interactively change the data type of a variable in table output. Right-click the variable column in the table, select **Convert from datatype to**, and select from the available options.

T = 5x3 table


	Temps	Dates	Stations
1	58	'2017-04-18'	S2
2	72	'2017-04-18'	S2
3	56	'2017-04-30'	S1
4	90	'2017-05-01'	S3
5	76	'2017-04-27'	S2

Convert from cell to

- String
- Categorical
- Datetime
  - dd-MMM-yyyy HH:mm:ss
  - MM/dd/yy HH:mm:ss
  - MM/dd/yyyy HH:mm:ss
  - + 12 more
- Duration
  - dd-MMM-yyyy

Custom Input Format

## Live Editor Functions: Automatically convert selected code to a function

Break large scripts or functions into smaller pieces by converting selected code into functions in files or local functions. With one or more lines of code selected, on the **Live Editor** tab, in the **Code** section, click  **Refactor**, and then select from the available options. MATLAB creates a function with the selected code and replaces the original code with a call to the newly created function.

## MATLAB Online: Share folders and collaborate with others

Share your folders with a view-only link, or invite individual collaborators and set their editing permissions. Invitations can be accepted or declined.

After a folder is shared, you can manage the permissions of invited members, rescind invitations, or send additional invitations at any time.

## Projects: Organize, manage, and share your work using projects

Create projects in MATLAB to organize and share your work with others. Use projects to find files required to run your code, manage and share files and settings, and interact with source control.

To create a project from an existing folder of files, in the desired folder, go to the **Home** tab, and select **New** > **Project** > **From Folder**. MATLAB creates the project and adds your existing files to the project.

## MATLAB Startup: Execute MATLAB script or function non-interactively

To call a MATLAB script or function non-interactively, start MATLAB with the `-batch` option. The option is for non-interactive use in both scripting and command-line workflows. MathWorks recommends that you use the `-batch` option instead of the `-r statement` option for these cases.

For example, to run unit tests you created for your programs, from the operating system command prompt, type:

```
matlab -batch runtests
```

MATLAB:

- Starts without the desktop
- Does not display the splash screen
- Executes the `runtests` function
- Logs text to `stdout` and `stderr`
- Exits automatically with status

To test if a session of MATLAB is running in batch mode, call the `batchStartupOptionUsed` function.

For more information, see `matlab` (Windows), `matlab` (macOS), or `matlab` (Linux).

### **Toolbox Packaging: Install required add-ons with custom toolboxes**

When creating a custom toolbox, MATLAB detects the add-ons required by the toolbox. When someone installs your toolbox, the toolbox also downloads and installs the required add-ons.

# Language and Programming

## **append Function: Combine strings**

Combine text in string arrays, character vectors, and cell arrays of character vectors using the `append` function.

Unlike the `strcat` function, `append` treats all input data types the same. For example, if an input argument has trailing whitespace characters, then `append` always keeps them, even when the argument is a character vector.

## **MException class: Provide a suggested fix for an uncaught exception**

Provide a suggested fix, using the `Correction` class, for an exception when it is thrown and not caught. Use the `addCorrection` method to add the correction to an `MException` object.

## **Functionality being removed or changed**

### **Folders named resources are not allowed on the MATLAB path**

*Warns*

Starting in R2019a, the `resources` folder is a reserved folder, and folders with the name `resources` are not allowed on the MATLAB path. In previous releases, these folders were allowed on the MATLAB path.

If a folder named `resources` is specified when calling the `addpath`, `userpath`, or `pathdef` functions, MATLAB returns a warning and the folder is not added to the path. If you have a folder named `resources`, MATLAB is unable to run any of the contents of that folder, even if the `resources` folder is the current folder.

Rename all folders on the path named `resources`, and move any files you want to run in MATLAB out of folders named `resources`.

### **Cell array expansion is consistent with general array expansion**

*Behavior change*

Starting in R2019a, the dimensions of an expanded cell array are consistent whether you use curly braces or parentheses for indices. Previously, the output dimensions were different when you did not specify indices for all dimensions. Indexing with curly braces now matches the previous behavior for indexing with parentheses, which is consistent with general array expansion.

For more information, see the Compatibility Considerations section of `cell`.

### **Structure array expansion is consistent with general array expansion**

*Behavior change*

Starting in R2019a, the dimensions of an expanded structure array are consistent whether you assign a value to a single field using dot notation or assign an entire structure to the array. Previously, the output dimensions were different when you did not specify indices for all dimensions. Assigning to a field using dot notation now matches the previous behavior of assigning a structure, which is consistent with general array expansion.

For more information, see the Compatibility Considerations section of `struct`.

### **Class properties using size validation no longer unconditionally reshape empty arrays**

#### *Behavior change*

In previous releases, if a class defined a property using a size validation that contained unrestricted dimensions (indicated by a colon, such as `(:, :)`), then assigning an empty array of any size to the property resulted in an empty array of size `(0, 0)`. For example, given this class definition:

```
classdef MyClass
    properties
        Prop1(:, :)
        Prop2
    end
end
```

Assigning an empty array of any dimension to `Prop1` always resulted in an empty array of dimensions `(0, 0)`.

```
obj = MyClass;
obj.Prop1 = double.empty(0,5);
size(obj.Prop1)
```

```
ans =
     0     0
```

Assigning an empty array to `Prop2` produces the correct result because size validation with unrestricted dimensions is not used in the class.

```
obj = MyClass;
obj.Prop2 = double.empty(0,5);
size(obj.Prop2)
```

```
ans =
     0     5
```

Starting in R2019a, using unrestricted size validation for properties does not cause the size of empty arrays assigned to the properties to be reshaped to `(0, 0)`. In R2019a, the same class definition produces these results for assignment to `Prop1`.

```
obj = MyClass;
obj.Prop1 = double.empty(0,5);
size(obj.Prop1)
```

```
ans =
     0     5
```

### **Defining classes and packages using `schema.m` will not be supported in a future release**

#### *Still runs*

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

### **First argument to `ismethod` must be an object**

#### *Behavior change in future release*



The `ismethod` function is documented to look for a method of the object that is specified as the first input. However, the `ismethod` function treats `string` and `char` inputs as a class name and looks for the specified method in that class. Therefore, you cannot use `ismethod` to find a method of an input object that is a `string` or `char` array. In future releases, `ismethod` will return true only if the second input is the name of a method of the first input object. `ismethod` will not treat the first input as a class name.

For code that uses `ismethod` with a class name specified as a string scalar or character vector, you can substitute this expression as an alternative that will work in current and future versions.

```
any(strcmp('methodName', methods('ClassName')))
```

### **Program files larger than 128 MB or with high complexity will not be supported**

*Behavior change in future release*

In a future release, running or opening program files larger than approximately 128MB will not be supported. For files that contain only code (for example, `.m` and `.p` files), this limit will affect the file size. For files that store more than just code (for example, `.mlx` files), it will affect the size of the code. Running statements larger than 128MB, either directly in the Command Window or using the `eval` function, also will not be supported. In addition, code with high levels of complexity, such as a large number of deeply nested `if` statements, will not be supported. Currently, these files and code are supported but can cause errors or unpredictable behavior.

Code that is too large or complex will not run or open in MATLAB, and MATLAB will display an error.

Large program file or statement sizes often occur when using large portions of code (for example, over 500 lines) to define variables with constant values. To decrease the size of these files, consider defining the variables and saving them in a data file (for example, a MAT-file or `.csv` file). Then you can load the variables instead of executing code to generate them. This not only decreases the file size of your program, but can also increase performance.

## Data Analysis

### **xcorr and xcov Functions: Compute cross-correlation and cross-covariance in core MATLAB**

You can now compute the cross-correlation and cross-covariance of data using MATLAB. Previously, `xcorr` and `xcov` were only available in the Signal Processing Toolbox.

### **detrend Function: Remove piecewise polynomial trends, set continuity requirements, and specify sample points**

The `detrend` function now offers additional functionality.

- In addition to the constant and linear methods for removing piecewise trends, you can specify higher degree polynomials. For example, `detrend(A,3)` removes a cubic trend from the data in `A`.
- When supplying break points, you can use the `'Continuous'` parameter to specify whether the fitted trend must be continuous.
- The `'SamplePoints'` parameter allows you to define the sample points associated with the input data.

### **groupcounts Function: Count the number of group elements for arrays, tables, and timetables**

To count the number of elements in a group, use the `groupcounts` function.

### **grouptransform Function: Transform array data by group**

In addition to tables and timetables, you can now transform data in an array by group using the `grouptransform` function.

### **filloutliers, isoutlier, and rmoutliers Functions: Detect outliers using percentiles**

The `filloutliers`, `isoutlier`, and `rmoutliers` functions now offer Winsorization for detecting outliers using the `'percentiles'` option.

### **fillmissing and filloutliers Functions: Fill missing and outlier data using modified Akima interpolation**

You can now fill missing and outlier data with modified Akima interpolation using the `'makima'` option in the `fillmissing` and `filloutliers` functions.

### **fillmissing Function: Specify missing value locations**

To specify the locations of missing data when using the `fillmissing` function, use the `'MissingLocations'` parameter.

## min and max Functions: Return index information when operating on more than one dimension and specify linear indices

When simultaneously operating on more than one dimension with the `min` and `max` functions, you can now return index information corresponding to the minimum and maximum values.

You can also return the linear indices corresponding to the minimum and maximum values of the input array using the `'linear'` option.

## tall Arrays: Write custom sliding-window algorithms to operate on tall arrays

The functions `matlab.tall.movingWindow` and `matlab.tall.blockMovingWindow` enable you to write custom algorithms for sliding-window functions to operate on tall arrays.

## tall Arrays: Operate on tall arrays with more functions, including groupcounts, intersect, and svd

The functions listed in this table now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

<code>addvars</code>	<code>movevars</code>
<code>cospi</code>	<code>normalize</code>
<code>groupcounts</code>	<code>removevars</code>
<code>grouptransform</code>	<code>sinpi</code>
<code>inner2outer</code>	<code>splitvars</code>
<code>intersect</code>	<code>svd</code>
<code>mergevars</code>	<code>union</code>

In addition, some functions have removed limitations with tall arrays.

Functions	Added Support
<code>stack</code>	The two-output syntax <code>[S,iu] = stack(___)</code> now supports tall arrays. The second output returns indices that describe the mapping of rows in the stacking operation.  <i>Previously, multiple outputs were not supported for tall arrays.</i>
<code>groupsummary</code>	Grouped calculations on tall matrices and arrays are now supported.  <i>Previously, the first input was required to be a tall table or tall timetable.</i>

## Functionality Being Removed or Changed

### Default random number generator change for `tallrng`

#### *Behavior change*

Starting in R2019a, the default random number generator for `tallrng` is `threefry`. This generator offers performance enhancements for parallel calculations over the old default. In releases up to R2018b, the default random number generator for `tallrng` was `combRecursive`.

With a different default generator, MATLAB will generate different sequences of random numbers by default in the context of tall arrays. However, the statistics of these calculations will remain unaffected. Therefore, you should update any code that relies on the *specific* random numbers being generated. However, most calculations on the random numbers should be unaffected.

To set the generator to the settings used by default in R2018b and earlier releases, use the command:

```
tallrng(0, 'combRecursive')
```

## Data Import and Export

### **readmatrix, readvars, and readcell Functions: Read tabular data as a matrix, variables, or a cell array**

Read column-oriented data from text or spreadsheet files into a matrix, variables, or a cell array.

- `readmatrix` — Read homogeneous column-oriented data into a matrix.
- `readvars` — Read column-oriented data into variables. Each variable corresponds to a column of data in the file.
- `readcell` — Read heterogeneous data into a cell array.

### **writematrix and writecell functions: Write tabular data from a matrix or cell array to a text or spreadsheet file**

Write data from a matrix or a cell array to a text or spreadsheet file.

- `writematrix` — Write a homogeneous array to a file.
- `writecell` — Write a cell array to a file.

### **readtimetable and writetimetable Functions: Read and write timetables**

Read and write timetables in MATLAB.

- Use the `readtimetable` function to read timetables from text or spreadsheet files.
- Use the `writetimetable` function to write timetables to text or spreadsheet files.

### **detectImportOptions Function: Improve detection of import options for text and spreadsheet files**

Improve the detection of import options for text and spreadsheet files by passing additional information to the `detectImportOptions` function using these name-value pairs.

- `'ThousandsSeparator'` — Character separating thousands groups (numeric variables only)
- `'DecimalSeparator'` — Character separating integer part from fractional part (numeric variables only)
- `'TrimNonNumeric'` — Remove non-numeric characters from numeric variables (numeric variables only)
- `'ConsecutiveDelimitersRule'` — Procedure to handle consecutive delimiters (text files only)
- `'LeadingDelimitersRule'` — Procedure to handle leading delimiters (text files only)
- `'TreatAsMissing'` — Text to interpret as missing data (text files only)
- `'ReadRowNames'` — Read first column as row names
- `'ReadVariableNames'` — Read first row as variable names

For more information, see the `setvaropts` and `detectImportOptions` reference pages.

## **parquetread, parquetwrite, and parquetinfo Functions: Read, write, and get information from Parquet files**

Import and export column-oriented data from Parquet files in MATLAB. Parquet is a columnar storage format that supports efficient compression and encoding schemes. To work with the Parquet file format, use these functions.

- `parquetread` — Read columnar data from a Parquet file.
- `parquetwrite` — Write columnar data to a Parquet file.
- `parquetinfo` — Get information about a Parquet file.

For more information on the Parquet file format, see <https://parquet.apache.org/>.

## **write Function: Write tall arrays to Parquet files**

The `write` function now supports writing tall arrays to Parquet files. To write a tall array, set the `FileType` parameter to `'parquet'`, for example:

```
write('C:\myData',tX,'FileType','parquet')
```

## **Import Tool: Generate improved code when importing from text files**

**Import Tool** now functions consistently across different platforms and generates code that is easy to read for importing text files. For more information, see [Import Text File Data Using Import Tool](#).

## **thingSpeakRead and thingSpeakWrite Functions: Read or write data to the ThingSpeak IoT platform**

Access IoT data in ThingSpeak™ channels:

- Use `thingSpeakRead` to read data from ThingSpeak channels.
- Use `thingSpeakWrite` to write data to ThingSpeak channels.

For more information on the ThingSpeak platform, see <https://thingspeak.com/>.

## **writetable and imwrite Functions: Write to web-based storage services like Amazon Web Services and Azure Blob Storage**

Write tabular data and image files to remote locations using the `writetable` and `imwrite` functions. When writing data to remote locations, you must specify the full path using a uniform resource locator (URL). For example, write a csv file and a jpg file to Amazon S3 Cloud:

```
writetable(T,'s3://bucketname/path_to_file/my_text_file.csv');  
imwrite(I,'s3://bucketname/path_to_file/my_image.jpg');
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

## **ParquetDatastore Object: Create a datastore for a collection of Parquet files**

Read a collection of Parquet files into MATLAB workspace using `parquetDatastore`.

For more information on the Parquet file format, see <https://parquet.apache.org/>.

## **ImageDatastore Object: Create a subset of an existing datastore**

Create a subset of an image datastore using the `subset` method.

## **DsFileSet Object: Create a subset of a file collection**

You can create a subset of a `DsFileSet` object by using the `subset` method. The `DsFileSet` object helps you manage the iterative processing of large collections of files.

## **FileDatastore Object: Read large files by importing the file in smaller portions**

Read and process large files in smaller portions. For example, you can create a datastore that reads one array at a time from a large MAT-file that does not fit in the available memory. To set up your datastore to perform partial reads, use these name-value pairs: `'ReadMode'`, `'PreviewFcn'`, and `'BlockSize'`.

For more information, see `fileDatastore`.

## **Datastores: Combine and transform datastores**

Perform `combine` and `transform` operations on existing datastores.

- `combine` — Combine two or more datastores and return a new datastore representing the underlying datastores.
- `transform` — Transform an input datastore by using a specified transformation function and return the transformed datastore.

## **Custom Datastore: Read Hadoop based data from files, databases, and other non-file-based locations**

Author a custom datastore to access data stored in files or non-file-based data sources such as a databases using `matlab.io.datastore.HadoopLocationBased` mixin. Use this extension to specify the location of your data in Hadoop®. A custom datastore with the `HadoopLocationBased` mixin makes computations more efficient by leveraging the location of the data. With your custom datastore you can perform big data analysis by using `tall` arrays and `mapreduce`.

For more information on the custom datastore framework, see `Develop Custom Datastore`.

## **VideoReader function: Generate C and C++ code**

The `VideoReader` function supports C and C++ code generation using MATLAB Coder™.

## **ind2rgb function: Generate C and C++ code**

The `ind2rgb` function supports C and C++ code generation using MATLAB Coder.

## **Scientific File Format Libraries: NetCDF Library upgraded to version 4.6.1**

The NetCDF library is upgraded to version 4.6.1.

## **web function: Open external sites in system browser instead of MATLAB browser**

You can change the default behavior of the `web` function to open external sites in your system browser instead of the MATLAB browser. Using the system browser is recommended when opening external sites. To change the default behavior, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web**, and in the **System Web browser** section, select **Use system web browser when opening links to external sites (recommended)**.

## **Functionality being removed or changed**

### **NumberOfChannels property of the audioplayer and audiorecorder Objects is not recommended**

*Still runs*

The `NumberOfChannels` property of the `audioplayer` and `audiorecorder` objects is not recommended. Use the name `NumChannels` instead. To update your code, change instances of `NumberOfChannels` to `NumChannels`. The values of the properties are the same. There are no plans to remove the `NumberOfChannels` property at this time.

### **web Function**

*Behavior change in future release*

In future releases, the `web` function will open external sites using your system browser by default. Currently, the `web` function opens external sites using the MATLAB browser. Using the system browser is recommended when opening external sites.

To change the default browser, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web** and in the **System Web browser** section, select **Use system web browser when opening links to external sites (recommended)**.

### **hdftool is not recommended**

*Still runs*

In a future release, `hdftool` will be removed. To import HDF4 or HDF-EOS files, use the `hdfread` function instead.

### **csvread and csvwrite functions are not recommended**

*Still runs*

`csvread` and `csvwrite` are not recommended. Use `readmatrix` and `writematrix` instead. There are no plans to remove `csvread` and `csvwrite`.



This table shows typical usages of `csvread` and `csvwrite` and how to update your code to use `readmatrix` and `writematrix` instead.

Not Recommended	Recommended
<code>M = csvread(filename)</code>	<code>M = readmatrix(filename)</code>
<code>csvwrite('mydata.txt',M)</code>	<code>writematrix(M,'mydata.txt')</code>

For more information, see `readmatrix` and `writematrix`.

### **dlmread and dlmwrite functions are not recommended**

*Still runs*

`dlmread` and `dlmwrite` are not recommended. Use `readmatrix` and `writematrix` instead. There are no plans to remove `dlmread` and `dlmwrite`.

This table shows typical usages of `dlmread` and `dlmwrite` and how to update your code to use `readmatrix` and `writematrix` instead.

Not Recommended	Recommended
<code>M = dlmread(filename)</code>	<code>M = readmatrix(filename)</code>
<code>dlmwrite('mydata.txt',M)</code>	<code>writematrix(M,'mydata.txt')</code>

For more information, see `readmatrix` and `writematrix`.

### **xlsread and xlswrite functions are not recommended**

*Still runs*

`xlsread` and `xlswrite` are not recommended. Instead of `xlsread` and `xlswrite`:

- Use `readtable` and `writetable` for reading and writing mixed numeric and text data.
- Use `readmatrix` and `writematrix` for reading and writing homogeneous text or numeric data.
- Use `readcell` and `writetable` for reading and writing mixed numeric and text data.

There are no plans to remove `xlsread` and `xlswrite`.

This table shows typical usages of `xlsread` and `xlswrite` and how to update your code to use the recommended read and write functions.

Not Recommended	Recommended
Read spreadsheet data as a matrix using <code>xlsread</code> :  <code>M = xlsread(filename)</code>	Read spreadsheet data as a table:  <code>T = readtable(filename)</code>  However, to continue reading your data as a matrix, use:  <code>M = readmatrix(filename)</code>

Not Recommended	Recommended
Read spreadsheet data as a cell array using <code>xlsread</code> : <code>[~,~,C] = xlsread(filename)</code>	Import spreadsheet data as a table: <code>T = readtable(filename)</code> However, to continue importing your data as a cell array, use: <code>C = readcell(filename)</code>
Read a specific sheet and range as a matrix using <code>xlsread</code> : <code>M = xlsread(filename, sheet, range)</code>	Read a specific sheet and range as a table: <code>T = readtable(filename, 'Sheet', sheet, 'Range', range)</code> However, to continue reading your data as a matrix, use: <code>M = readmatrix(filename, 'Sheet', sheet, 'Range', range)</code>
Read a specific sheet and range as a cell array using <code>xlsread</code> : <code>[~,~,C] = xlsread(filename, sheet, range)</code>	Read a specific sheet and range as a table: <code>T = readtable(filename, 'Sheet', sheet, 'Range', range)</code> However, to continue reading your data as a cell array: <code>C = readcell(filename, 'Sheet', sheet, 'Range', range)</code>
Write tabular data to spreadsheets using <code>xlswrite</code> : <code>xlswrite(filename, M)</code>	To write tabular data to spreadsheets, use one of these options instead. Write a table: <code>writetable(T, filename)</code> Write a matrix: <code>writematrix(M, filename)</code> Write a cell array: <code>writecell(C, filename)</code>

For more information, see `readmatrix`, `writematrix`, `readcell`, `writecell`, `readtable`, and `writetable`.

### **HadoopFileBased is not recommended**

*Still runs*

`HadoopFileBased` is not recommended. Use `HadoopLocationBased` instead. There are no plans to remove `HadoopFileBased`.

Starting in R2019a, use the `HadoopLocationBased` mixin to add Hadoop support to your custom datastore. The `HadoopLocationBased` mixin provides support for non-file-based data where as `HadoopFileBased` supports file-based data only.

For more information on the custom datastore framework, see [Develop Custom Datastore](#).

## Mathematics

### Solve assignment problem with `matchpairs` and `equilibrate`

New functions enable you to solve the assignment problem in a variety of contexts.

- `matchpairs` — Create a linear mapping between the rows and columns of a cost matrix. This assigns rows to columns in such a way that the global cost is minimized.
- `equilibrate` — Permute and rescale a matrix  $A$  such that the new matrix  $B = R*P*A*C$  has only 1s and -1s on its diagonal, and all off-diagonal entries are not greater than 1 in magnitude. When computing a preconditioner to iteratively solve a linear system, use equilibration to improve the condition of a matrix and allow for improved preconditioners.

### `graph` and `digraph` Objects: Construct graphs with categorical nodes

The `graph`, `digraph`, and `addedge` functions now support categorical node names as inputs. This enables you to use data that is imported as `categorical` to create a graph, without the need for data type manipulation.

## Graphics

### parallelplot Function: Visualize tabular or matrix data with multiple columns by using a parallel coordinates plot

To create a parallel coordinates plot, use the `parallelplot` function. Rows of the input data correspond to lines in the plot, and columns of the input data correspond to coordinates in the plot. To group the lines in the plot, you can use either the `'GroupVariable'` name-value pair argument with tabular data or the `'GroupData'` name-value pair argument with matrix data.

### Data Tips: Pin and customize data tips in charts

The data tips that appear as you hover over a chart become persistent (pinned) when you click them. Clicking a second time unpins the data tip.

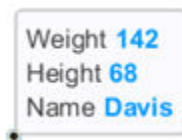
For some types of charts, you can customize the contents of the data tips. For example, you can edit the data tip labels, change the displayed values, or change the font size. Also, you can add or delete rows from the data tips. Charts that support these customizations include Scatter, Stair, Stem, Line, and Surface objects with a `DataTipTemplate` property.

- To edit the labels interactively, double-click a label, type the text you want, and then click outside the data tip. To make other customizations interactively, right-click the data tip and select **Edit Properties...** Use the fields in the Property Inspector that opens to make any changes.
- To customize the data tip programmatically, use the `DataTipTemplate` property of the chart object. For example, this code plots sample patient data from a table as a scatter chart. Then it changes the font size and labels of the data tips. For more information, see `DataTipTemplate`.

```
tbl = readtable('patients.xls');
s = scatter(tbl.Weight,tbl.Height);
s.DataTipTemplate.FontSize = 12;
s.DataTipTemplate.DataTipRows(1).Label = 'Weight';
s.DataTipTemplate.DataTipRows(2).Label = 'Height';
```

You can add a new row to the data tip using the `dataTipTextRow` function. For example, add a third row that shows the patient name from the table.

```
s.DataTipTemplate.DataTipRows(3) = dataTipTextRow('Name',tbl.LastName);
```



### Axes Interactions: Customize chart interactions such as dragging to pan or scrolling to zoom

Create a customized set of chart interactions by setting the `Interactions` property of the axes. These interactions are built into the axes and are available without having to select any buttons in the axes toolbar. Some types of interactions are enabled by default, depending on the content of the axes.

For more information, see [Control Chart Interactivity](#).

## Ruler Panning: Pan an axis to change its limits without having to use the pan tool

Drag an axis to change the limits along a dimension of a plot. This functionality is available for most Cartesian plots, even when the pan tool in the axes toolbar is disabled.

## Property Inspector: Navigate and control visibility of graphics objects interactively

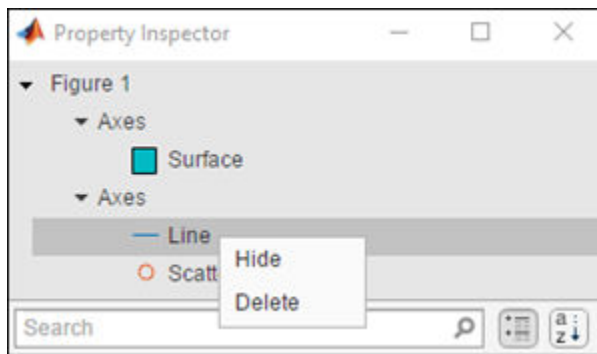
You can use the object browser that appears at the top of the **Property Inspector** to navigate and control the visibility of graphics objects. When you select an object using the object browser, the object appears selected in the figure and the properties appear in the inspector.

The object browser has a collapsed view and an expanded view.

- The collapsed view (default view) shows the currently selected object and its direct hierarchy. Click one of the object names to see its properties in the Property Inspector.

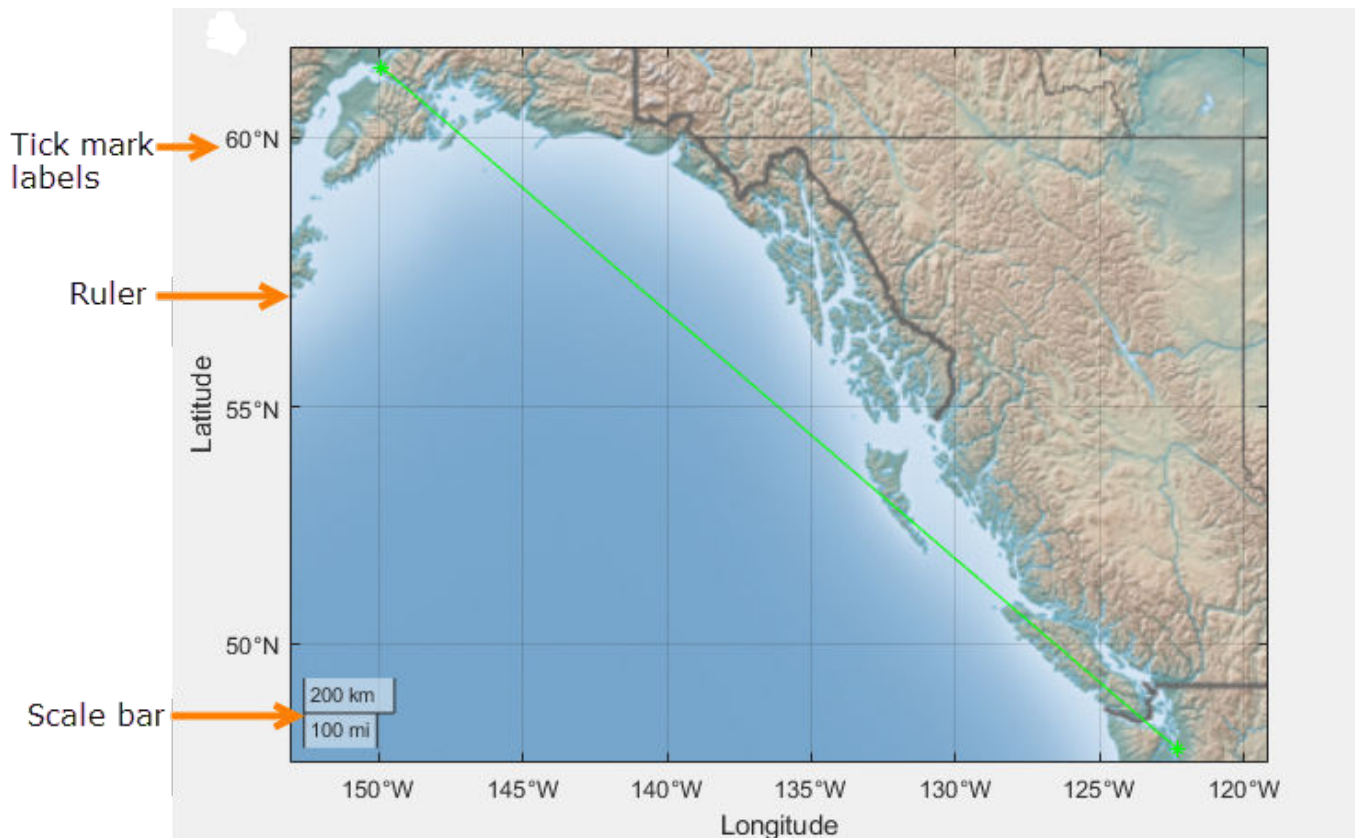


- The expanded view shows the graphics object hierarchy of the figure. Right-click an object name to show, hide, or delete the graphics object. Select multiple objects using **Ctrl**+click.



## Geographic Plots: Geographic rulers, scale bar, CurrentPoint, and ginput

Plots on geographic axes include customizable rulers and a scale bar.



Use the `geotickformat` function to customize rulers.

GeographicAxes support the `CurrentPoint` property. Use this property to get the current coordinates of the mouse pointer on a geographic axes.

## Graphics Export: Export axes with tighter cropping using the axes toolbar

Click or tap the export button in the axes toolbar to save the axes as an image or PDF file. The saved content is tightly cropped around the axes with minimal white space.



## Chart Resizing: Resize charts with improved layouts

The layout is improved when you resize a chart that can be a child of a figure (such as a heatmap). This automatic resizing behavior adjusts the font sizes and spacing between elements in the chart to provide the best possible presentation for the new size.

Changing the `FontSize` property of a chart disables the automatic resizing of the fonts.

## Colors Values: Specify colors using hexadecimal color codes

Specify hexadecimal color codes when setting the color of graphics objects. For example, `set(gca, 'XColor', '#FF8800')` sets the x-axis color to orange. Use either the six-digit or three-digit form to specify a color. The characters are not case-sensitive. Thus, '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

## Categorical Values: Specify categorical arrays for functions and objects that use lists of text

Many functions and object properties that use lists of text items now accept categorical arrays in addition to cell arrays and string arrays. When you specify a categorical array, MATLAB uses the values in the array, not the categories. Thus you might need to write additional code to extract the values you want to use. For example, this code finds the unique entries of the `cities` categorical array before setting the x-axis tick labels.

```
bar([10 20 30])
cities = categorical({'Boston', 'Boston', 'Berlin', 'Paris', 'Berlin'});
xticklabels(unique(cities))
```

See the documentation for a specific function or object to determine whether it accepts categorical values.

## rendererinfo Function: Get renderer information for any axes

Use the `rendererinfo` function to get information about the graphics renderer used for any axes or chart that can be a child of a figure.

## Version History

Use the `rendererinfo` function instead of the `opengl` function to get information about the graphics renderer.

## Functionality being removed or changed

### Using the `opengl` function to get information about the graphics renderer is not recommended

*Still runs*

Using the `opengl` function to get information about the graphics renderer is not recommended. Specifically, these syntaxes are not recommended:

- `opengl info`
- `d = opengl('data')`

There are no plans to remove support for these syntaxes at this time. Instead of calling `opengl` to get the renderer information, call the `rendererinfo` function instead:

```
info = rendererinfo(ax)
```

Specify `ax` as any type of axes or a chart that can be a child of a figure (such as a heatmap). The output is a structure containing most of the same information as the `opengl` function provides.

Fields in opengl Structure	Corresponding Fields in rendererinfo Structure
d.Version	info.Version
d.Vendor	info.Vendor
d.Renderer	info.RendererDevice
d.RendererDriverVersion	info.Details.RendererDriverVersion
d.RendererDriverReleaseDate	info.Details.RendererDriverReleaseDate
d.MaxTextureSize	info.Details.MaxTextureSize
d.Visual	No longer needed
d.Software	This information is stored in <code>info.GraphicsRenderer</code> , but to get the equivalent logical value, use <code>strcmp(info.GraphicsRenderer, 'OpenGL Software')</code>
d.HardwareSupportLevel	info.Details.HardwareSupportLevel
d.SupportsGraphicsSmoothing	info.Details.SupportsGraphicsSmoothing
d.SupportsDepthPeelTransparency	info.Details.SupportsDepthPeelTransparency
d.SupportsAlignVertexCenters	info.Details.SupportsAlignVertexCenters
d.Extensions	No longer needed
d.MaxFrameBufferSize	info.Details.MaxFrameBufferSize

### Heatmaps interpret text using TeX markup

#### *Behavior change*

Starting in R2019a, heatmaps created with the `heatmap` function interpret text using TeX markup instead of displaying the literal characters. If you want to use a TeX markup character in regular text, such as an underscore (`_`), then insert a backslash (`\`) before the character you want to include. The backslash is the TeX escape character. For more information on using TeX markup, see the `Interpreter` property of the text object.



## App Building

### **UIImage Function: Display an icon, logo, or picture in apps and on the App Designer canvas**

To display a picture, icon, or logo in your app, call the `UIImage` function programmatically or, in App Designer, drag and drop an image component from the **Component Library** onto the canvas.

Image components are supported only in App Designer apps and in figures created with the `UIImage` function.

### **UITable Function: Sort tables interactively when using table arrays**

To create tables that can be sorted interactively, use `Table` arrays and configure the `ColumnSortable` property of the `Table` object. Use the `DisplayData` property and a `DisplayDataChangedFcn` callback if you want to update your visualizations based on how a user sorts a table containing `Table` data.

### **Auto Resize: Automatically resize components when an app is made smaller**

When a parent container is resized smaller than its initial size, `AutoResizeChildren` now reduces the white space between components and shrinks the components themselves to maintain usability. For more information, see [Managing Resizable Apps in App Designer](#).


### **Scrolling Grids: Create apps with scrollable grids**

Enable interactive scrolling in your grid layout manager by setting the `Scrollable` property of the grid to `'on'`. See `uigridlayout` for more information.

### **App Designer: Create apps that automatically reflow content based on device size**

Create 2-panel or 3-panel preconfigured apps that automatically resize and reflow content based on screen size, screen orientation, and platform. Use apps with auto-reflow if you expect to run or share your apps across multiple environments or desktop resolutions. For more information, see [Apps with Auto-Reflow](#).

### **App Designer: Add and configure a grid layout manager on the App Designer canvas**

Structure the layout of your app by dragging a grid layout manager from the **Component Library** onto the canvas. To configure the grid layout in **Design View**, select the  icon from the upper-left hand corner of the grid, or right-click and select **Configure grid layout**. Then, select a row or column to edit. For more information, see the `uigridlayout` function or `GridLayout` Properties.

Grid layout managers are supported only in App Designer apps or in figures created with the `uifigure` function.

## App Designer: Rearrange the order of callbacks

To rearrange the order of callbacks, go to the **Code Browser**, select the callback you wish to move, and then drag the callback into a new position in the list. This also repositions the callback in the editor.

## App Designer: Create new apps using App Designer Start Page options

From the App Designer Start Page you can now do the following.

- Create a new blank app or a new responsive app with auto-reflow.
- Start a tutorial or access featured content from the top banner.
- View a list of your recent apps.
- Open apps from a file path.
- Select an example from the **Getting Started** or **Programming Tasks** example sections.

The Start Page appears when you launch App Designer. Once you are in the design environment, you can get back to the Start Page by selecting the New  icon from the **Designer** toolstrip.

## App Designer: Control font, code, and autosave settings using MATLAB Preferences

Control these settings through the Preferences dialog box

- Autocoding for patterns like parentheses, block endings, and comment wrapping
- Keyboard preferences for automatic code suggestions and completions
- Autosave preferences upon clicking away from a file
- Font size preference for App Designer **Code View**

Changes to the autosave and font size preferences apply to only the App Designer Editor. When you set the autocoding or keyboard preferences, the change applies to the MATLAB Editor and to App Designer.

## App Designer: Access context-sensitive help in Code View

To open the documentation for a component, function, or callback in your code, highlight the element and then press **F1**, or right-click and choose help for the code element you selected.

## App Designer: Zoom in App Designer

Hold **Ctrl** and move the scroll wheel in the App Designer window to zoom in or out. To return to the default scale, press **Ctrl+0**.

## Graphics Support: Explore data using axes toolbar and data tips in apps created with the uifigure function

Use the axes toolbar and data tips to explore plotted data interactively in App Designer apps and in figures created with the `uifigure` function. The axes toolbar and data tips are on by default for `axes` and `uiaxes` objects in a `uifigure`.

### Version History

- Axes toolbar — In previous releases, the axes toolbar was not enabled for `axes` or `uiaxes` objects in a `uifigure` object. Now, it is enabled by default. You can turn it off by setting the `Visible` property of the `AxesToolbar` object to `'off'`. For more information, see `AxesToolbar` Properties.
- Data tips — In previous releases, data tips were not enabled for `axes` or `uiaxes` objects in a `uifigure` object. Now, to control whether the axes interactions are enabled, use the `disableDefaultInteractivity` and `enableDefaultInteractivity` functions. For example,

```
uf = uifigure;  
ax = axes(uf);  
plot(ax, rand(5))  
disableDefaultInteractivity(ax)
```

## Deployed Web Apps: Share resizable apps or create apps that open web pages

In deployed web apps you can now do the following:

- Interactively resize your web app.
- Program your web app to open another URL using the `web` function.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

## MATLAB Online: Create and edit App Designer apps using MATLAB Online

Create or edit apps in MATLAB Online using the App Designer development environment (supported only for Google Chrome browsers).

## App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.hover` method enables you to perform hover gestures in tests on `axes`, `UI axes`, and `UI figure` objects. For more information, see the `hover` reference page.

## **App Testing Framework: Perform press gesture on axes, UI axes, and UI figures**

The `matlab.uitest.TestCase.press` method enables you to perform press gestures in tests on UI axes and UI figure objects. For more information, see the `press` reference page.

## **App Testing Framework: Perform type gesture on date picker objects**

The `matlab.uitest.TestCase.type` method enables you to perform type gestures on date picker objects. For more information, see the `type` reference page.

## **Functionality Being Removed or Changed**

### **javacomponent function and JavaFrame property will be removed in a future release**

*Warns*

The undocumented `javacomponent` function and `JavaFrame` property will be removed in a future release. The `JavaFrame` property still runs, but returns a warning. The `javacomponent` function still runs, without warning, but will begin to warn in an upcoming release. Users are encouraged to update their code to use documented alternatives. For a list of documented functionality you can use instead, see [Java Swing Alternatives for MATLAB Apps on mathworks.com](http://mathworks.com).

### **Support for running deployed web apps in Internet Explorer will be removed in a future release**

*Still runs*

Support for running deployed web apps in Internet Explorer will be removed in a future release. Use the current versions of Google Chrome (recommended), Safari, Firefox, or Microsoft Edge to run deployed web apps instead.

## Performance

### **MATLAB and Simulink startup on macOS platforms**

To prevent performance regression at startup for MATLAB and Simulink on macOS platforms, MathWorks recommends using MATLAB R2018b Update 4 or later with macOS 10.13.6 Update or later.

### **sortrows Function: Sort rows of large matrices faster**

For large matrices, you can now sort rows faster using the `sortrows` function.

For example, on a test system, this code runs faster in R2019a than in previous releases.

```
A = repmat(eye(200),500,1);
sortrows(A)
```

### **uitable Function: Faster performance using table arrays**

Tables created with the `uitable` function and with data specified as a `table` array have better rendering performance and higher frame rates while scrolling. Tables that use `table` arrays render up to 40% faster, and interaction performance (like scrolling) is up to 75% faster. For example, on a test system, this code renders the tables faster in R2019a than in previous releases.

```
rows = 10000;
columns = 25;
data = array2table(randi(30, [rows, columns]));
fig = uifigure;
tbl = uitable(fig, 'Data', data);
```

For more information about using `table` arrays in Table UI components, see [Table Array Data Types in App Designer Apps](#).

## Software Development Tools

### **checkcode Function: Get the modified cyclomatic complexity of functions**

Use the `checkcode` function with the `'modcyc'` option to get the modified cyclomatic complexity of each function in a file. The modified cyclomatic complexity for a function is equal to the McCabe complexity except for one difference. McCabe complexity counts each individual case within a `switch` statement as 1, while modified cyclomatic complexity counts the entire `switch` statement as 1. In general, `switch` statements are simpler than nested `if-elseif-else` statements and therefore, the modified cyclomatic complexity is often considered a better measure of code complexity.

### **Source Control Integration: Synchronise MATLAB Git status with external Git clients**

If you use an external Git client, MATLAB now listens to external changes to working copies of `.git` folders and refreshes the file status if needed. MATLAB keeps the Git file status in sync when using another Git client, both in the MATLAB current folder and in a project.

### **Unit Testing Framework: Display code coverage metrics in HTML format**

The `matlab.unittest.plugins.codecoverage.CoverageReport` class provides an HTML code coverage report format to display code coverage metrics. Use this format with `matlab.unittest.plugins.CodeCoveragePlugin` to produce the report.

### **Unit Testing Framework: Specify sources for collections of code coverage data with `runtests`**

The `runtests` function enables you to specify the source code files to include in the code coverage report. Use the `ReportCoverageFor` name-value input to specify the files or folders containing source files to include in the tests.

### **Unit Testing Framework: `runperf` collects more samples to achieve its target margin of error**

The default maximum number of sample measurements that `runperf` makes when running performance measurements has increased to 256. Specify the number of sample measurements using the `matlab.perftest.TimeExperiment.limitingSamplingError` method.

### **Unit Testing Framework: Return performance test results as `TimeResult` arrays**

The `runperf` function now returns a `matlab.perftest.TimeResult` array containing the results of the specified performance tests. This class derives from the `matlab.unittest.measurement.MeasurementResult` class, which is now an abstract class.

## Unit Testing Framework: Load previously saved MeasurementResult objects as DefaultMeasurementResult

MeasurementResult objects saved in previous releases are loaded as matlab.unittest.measurement.DefaultMeasurementResult objects. This class is derived from the MeasurementResult class.

## Unit Testing Framework: Use matlab.unittest.fixtures.Fixture.onFailure method only in subclasses

The onFailure method now has protected access. In previous releases, onFailure had public access. This change better supports the use of onFailure to produce additional diagnostics in case of a failure during fixture setup or teardown in classes derived from Fixture.

## Unit Testing Framework: Compare tables that contain no rows

In previous releases, for tables that had no rows (that is, that had a first size dimension of zero), the matlab.unittest.constraints.IsEqualTo constraint did not compare the table column variables when determining equality. Now, IsEqualTo always compares the size and type of each column variable.

For example, comparing these two tables fails now because the column variables are different types (double and cell).

```
tc = matlab.unittest.TestCase.forInteractiveUse;
a = table(zeros(0,2));
b = table({});
tc.verifyEqual(a,b)
```

Verification failed.

```
-----
Framework Diagnostic:
-----
verifyEqual failed.
--> Path to failure: <Value>.Var1
    --> Classes do not match.

    Actual Class:
           double
    Expected Class:
           cell

    Actual double:
           0x2 empty double matrix
    Expected cell:
           0x0 empty cell array

    Actual Value:
           0x1 empty table
    Expected Value:
           0x1 empty table
```

## Unit Testing Framework: Create test suite array from tests in project

The `fromProject` method enables you to create a test suite array from the files in a project that are labeled with the `Test` classification. For more information, see the `matlab.unittest.TestSuite.fromProject` reference page.

## Unit Testing Framework: Run tests from files in project using `runtests` or `testsuite`

Run test files from projects using the `runtests` or `testsuite` functions. The `IncludeReferenceProjects` name-value pair argument enables you to include in the test suite files from a project that are labeled with the `Test` classification.

## Unit Testing Framework: Specify verbosity enumeration as a string or character vector

You can specify the verbosity level argument for the following methods as a string scalar or character vector that correspond to the `matlab.unittest.Verbosity` enumeration member name.

- `matlab.unittest.TestCase.log`
- `matlab.unittest.TestRunner.withTextOutput`
- `matlab.unittest.plugins.TestRunProgressPlugin`
- `matlab.unittest.plugins.LoggingPlugin.withVerbosity`
- `matlab.unittest.plugins.DiagnosticsOutputPlugin`

## App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.hover` method enables you to perform hover gestures in tests on axes, UI axes, and UI figure objects. For more information, see `hover`.

## App Testing Framework: Perform press gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.press` method enables you to perform press gestures in tests on UI axes and UI figure objects. For more information, see `press`.

## App Testing Framework: Perform type gesture on date picker objects

The `matlab.uitest.TestCase.type` method enables you to perform type gestures on date picker objects. For more information, see `type`.

## Mocking Framework: Create mocks for classes that use custom metaclasses

The unit testing framework can now create mocks for classes that use custom metaclasses to define custom class, property, method, and event attributes.



## Mocking Framework: Create mocks for classes that use property validation

The unit testing framework can now create mocks for classes that use property validation. For information on property validation, see [Validate Property Values](#).

## Mocking Framework: Specify which methods to mock

When creating a mock object, you can control which methods are mocked in the test case. Use the `createMock` method with the `MockMethods` name-value pair argument to specify the method to mock. This feature enables tests to mock only those methods that are important to the test case, which can improve performance when superclasses define many methods.

## Functionality being removed or changed

### **matlab.unittest.fixtures.Fixture.onFailure method has protected access**

*Behavior change*

In release R2019a, the `onFailure` method `Access` attribute is changed from `public` to `protected`. This change restricts the use of `onFailure` to classes derived from `Fixture`.

### **matlab.unittest.constraints.IsEqualTo always compares table column variables**

*Behavior change*

In release R2019a, the `IsEqualTo` constraint always compares the size and type of column variables.

## External Language Interfaces

### **C++: Use C++ classes from third-party libraries in MATLAB**

If you have a library that exports C++ constructs, including classes, functions and enumerations, then you can use this functionality directly in MATLAB. For more information, see [C++ Libraries](#).

If you have a C shared library, then use the `loadLibrary` function as described in [C Libraries](#).

### **Python: Version 3.7 support**

MATLAB now supports CPython 3.7, in addition to existing support for 2.7, 3.5, and 3.6.

For more information, see [Install Supported Python Implementation](#).

### **Version History**

To start the MATLAB engine asynchronously from Python 3.7, use the (`background=True`) keyword argument for `matlab.engine.start_matlab`. To call a MATLAB function asynchronously, use the `background=True` keyword argument for `matlab.engine.MatlabEngine`. Do not use the `async` argument for either function, since it is a keyword in Python 3.7. You also can use the `background` argument for all supported versions of Python.

### **Python engine: Data type support**

The Python engine now supports this functionality:

- Convert MATLAB strings to Python strings
- Pass function handles to Python with the `feval` command
- Pass MATLAB value objects as opaque objects

### **C++ MEX: Execute MEX function out of process**

Run C++ MEX functions in processes that are separate from the MATLAB process. You can run multiple MEX functions in the same process and can create multiple processes to execute MEX functions. For more information, see [Out-of-Process Execution of C++ MEX Functions](#).

### **MEX functions: Use customer version of Boost library**

Although MATLAB builds with Boost library version 1.56.0, as of MATLAB R2018a, you can use any Boost library version in a MEX function.

### **MATLAB Data Array: Support for row-major memory layout**

Create a `matlab::data::Array` with data memory layout specified as column-major (default) or row-major. For more information, see the `memoryLayout` parameter in `createArrayFromBuffer`. To determine the memory layout for an existing `matlab::data::Array`, call `getMemoryLayout`.

## Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2019 with Microsoft Visual Studio 2015 and 2017 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2019 for Fortran	macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see [Supported and Compatible Compilers](#).

## Hardware Support

### **MATLAB Support Package for Parrot Drones: Control Parrot Mambo FPV drone from MATLAB and acquire sensor data**

The MATLAB Support Package for Parrot® Drones is available from release R2019a onwards.

The support package includes functions to pilot a Parrot Mambo FPV drone by sending MATLAB commands to control its direction, speed, and orientation. You can also read the flight navigation data such as speed, height, and orientation using MATLAB commands.

### **Deploy Sense HAT functions on Raspberry Pi hardware**

These Sense HAT functions from the MATLAB Support Package for Raspberry Pi Hardware are enhanced to generate code: `sensehat`, `readHumidity`, `readPressure`, `readTemperature`, `readAngularVelocity`, `readAcceleration`, `readMagneticField`, `readJoystick`, `displayImage`, `writePixel`, and `clearLEDMatrix`. You can now deploy these functions on the hardware.

### **Functionality being changed or removed**

#### **The `i2cdev` and `spidev` functions will be removed in a future release**

*Warns*

Use `device` instead of `i2cdev` and `spidev` to connect to I2C or SPI devices on Arduino hardware.

#### **The property `Pins` of `servo` object will be removed in a future release**

*Warns*

Use the property `Pin` instead of `Pins` to get the pin number of the Arduino hardware and the Adafruit Motor Shield V2 for Arduino hardware to which the servo motor is connected. For more information, see [Connection to servo motor on Arduino](#) and [Connection to servo motor on Adafruit Motor Shield V2](#).

#### **The class `arduinoio.LibraryBase` will be removed in a future release**

*Warns*

Use the class `matlabshared.addon.LibraryBase` instead of `arduinoio.LibraryBase` for deriving Arduino add-on libraries.

#### **MATLAB support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in a future release**

*Warns*

The support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in a future release

#### **MATLAB support for Arduino hardware boards has been removed**

*Errors*

These Arduino hardware boards are no longer supported:

- Arduino Fio
- Arduino Mini
- Arduino Pro



# R2018b

---

**Version: 9.5**

**New Features**

**Bug Fixes**

**Version History**


## Desktop

### Live Editor: Organize live scripts using additional subheading styles

Format text in live scripts using the new **Heading 2** and **Heading 3** text styles. To apply a text style, go to the **Live Editor** tab and in the **Text** section, select any of the options under the **Text Style** dropdown.

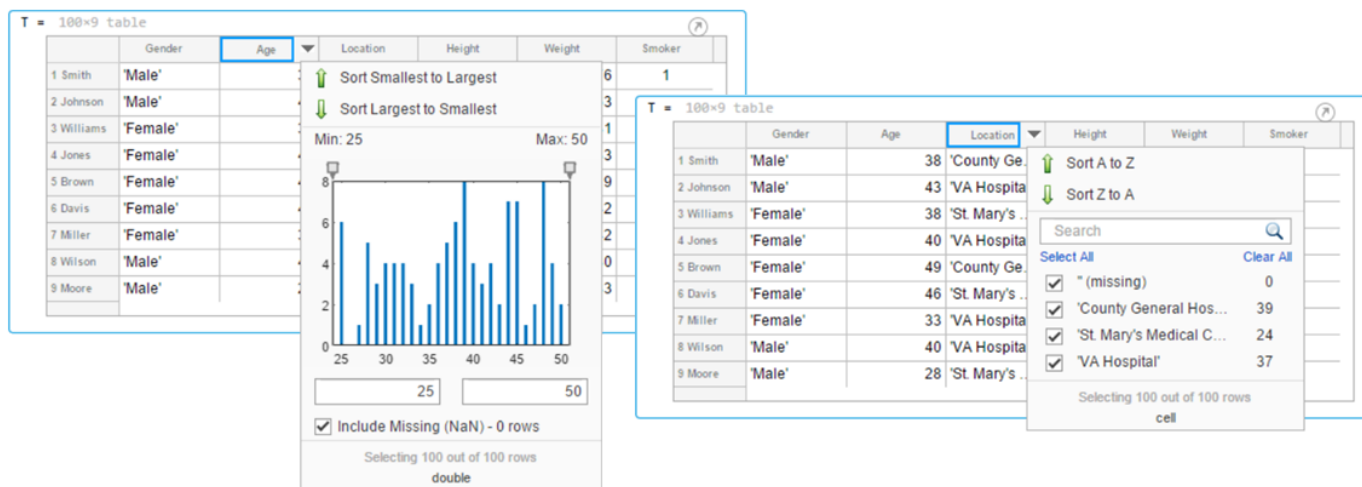
For more information, see Format Files in the Live Editor.

### Live Editor: Navigate within a live script using internal hyperlinks

Use internal hyperlinks to navigate to locations within a live script. To insert an internal hyperlink, go to the **Insert** tab and click  **Hyperlink**. Enter your display text, select **Internal Hyperlink**, and then click anywhere in the document to select the target.

### Live Editor: Filter table output interactively, and then add the generated code to the live script

In the Live Editor, you can filter table data interactively. To filter data in a table, click the down arrow ▼ to the right of a variable name in the table and select from the available filtering options.



The screenshot displays two overlapping table views in the Live Editor. The left table has 'Age' selected, showing sorting options (Sort Smallest to Largest, Sort Largest to Smallest) and a histogram of the data. The right table has 'Location' selected, showing sorting options (Sort A to Z, Sort Z to A) and a filter dropdown menu. The filter menu includes a search bar, 'Select All', 'Clear All', and a list of location categories with counts: '(missing)' (0), 'County General Hos...' (39), 'St. Mary's Medical C...' (24), and 'VA Hospital' (37). A status bar at the bottom of the filter menu indicates 'Selecting 100 out of 100 rows cell'.

To add the generated code to your live script, use the **Update Code** button below the table. Adding the generated code to your live script ensures that the sorting is reproduced the next time you run the live script.

### Live Editor: Create new and open existing live scripts faster

New and existing live scripts open faster than in previous releases.

### Live Editor: Change case of text or code

In the Live Editor, you can change the case of selected text or code from all uppercase to lowercase, or vice versa. To change the case, select the text, right-click, and select **Change Case**. You also can






press **Ctrl+Shift+A**. If the text contains both uppercase and lowercase text, MATLAB changes the case to all uppercase.

In MATLAB Online, this feature also is available in the Editor.

## Comparison Tool: Merge two versions of a live script or function

When comparing live scripts or live functions using the Comparison Tool, you can merge changes from one file to the other. Merging changes can be useful when resolving conflicts between different versions of a file.

To merge two live scripts or functions, go to the **Live Editor** tab and in the **File** section, click **Compare**. A new window opens and displays the two files side by side. Select the  **Merge Mode** button to start the merge.



Use the  button to replace content in the right pane with content from the left pane. The right pane contains the merged result. To save the result, click  **Save Result**.

For more information, see Compare and Merge Live Code.

## Add-On Manager: Install and manage multiple versions of a custom toolbox

You can install multiple versions of a custom toolbox in the Add-On Manager. Having multiple versions of a custom toolbox installed is useful if you regularly use multiple versions and switch between them.

To install an additional version of a custom toolbox without overwriting any of the other installed versions, use the `matlab.addons.install` function and specify 'add' as the installation option. For example, `matlab.addons.install('C:\myAddons\GUI Layout Toolbox 2.1.2.mltbx', 'add')`.

To select which version of the toolbox is enabled, go to the **Home** tab and select  **Add-Ons > Manage Add-Ons**. Click the  button to the right of the toolbox you want. Then in the **Versions** menu, select from the available versions. Selecting a version enables that version and disables all other installed versions of the toolbox. You also can use the `matlab.addons.enableAddon` function.

## Add-On Manager: Save add-ons to new default location

MATLAB now saves add-ons to a new default location. The default location is platform-specific.

- Windows platforms — `C:\Users\username\AppData\Roaming\MathWorks\MATLAB Add-Ons`.
- Linux platforms — `~/MATLAB Add-Ons`.
- Mac platforms — `~/Library/Application Support/MathWorks/MATLAB Add-Ons`.

For more information, see Manage Your Add-Ons.

## **Documentation: View MATLAB documentation in Spanish**

A *subset* of MATLAB documentation in Spanish is available on the web to licensed MATLAB users. For more information, see Translated Documentation.

## Language and Programming

### **string Arrays: Use string arrays in MATLAB, Simulink, and Stateflow**

Specify text as string arrays where you previously specified text as character vectors or cell arrays of character vectors. Use string arrays for data, properties, and name-value pair arguments. Specify strings using double quotes, just as you specify character vectors using single quotes.

For more information on string arrays, see Characters and Strings. For guidelines on accepting strings in your own code, see Update Your Code to Accept Strings.

MathWorks encourages the use of string arrays. For backward compatibility, MathWorks products will continue to support the use of character vectors and cell arrays of character vectors.

### **convertContainedStringsToChars Function: Convert string arrays at any level of cell array or structure**

To make your existing code accept string arrays, or cell arrays and structure arrays that contain strings, use the `convertContainedStringsToChars` function on the entire input argument list. For more information on accepting strings in your own code, see Update Your Code to Accept Strings.

### **Enumerations: Improved performance of set operations with enumerations**

When called with enumeration arrays, execution of set operation functions such as `ismember` is faster.

### **WSDL Web Services Documents: Required Tools Update**

As of MATLAB R2018b, the supported versions of the Oracle® Java JDK and the Apache CXF programs that are required to use a WSDL Web service in MATLAB have changed.

For more informations, see Set Up WSDL Tools.

### **Version History**

Download and install the JDK software from the Java SE Downloads Web page <https://www.oracle.com/technetwork/java/javase/downloads>. Choose the Java SE Development Kit 8.

Download the latest version 3.2 release of the Apache CXF tool from <https://cxf.apache.org/download>.

### **Functionality being removed or changed**

#### **validateattributes check for 'finite' and 'nonnan' attributes**

*Behavior change in future release*

In the `validateattributes` function, the 'finite' and 'nonnan' attributes no longer require that the input passes an `isnumeric` check.

**Folders named resources will not be allowed on the MATLAB path***Still runs*

In future releases, the `resources` folder will become a reserved folder, and folders with the name `resources` will not be allowed on the MATLAB path. Currently, these folders are allowed on the MATLAB path.

If a folder named `resources` is specified when calling the `addpath`, `userpath`, or `pathdef` functions, MATLAB will return a warning and the folder will not be added to the path.

Rename all folders on the path named `resources`.

## Mathematics

### boundaryshape Function: Create a polyshape object from a 2-D triangulation

You now can use the boundaryshape function to convert a 2-D triangulation object to a polyshape object.

### polyshape Objects: Specify when to keep collinear points when creating a polyshape

When creating a polyshape object, collinear points are removed by default. These functions now offer the option to keep collinear points as vertices of the returned polyshape using the name-value pair 'KeepCollinearPoints'.

polyshape	subtract
addboundary	union
intersect	xor
simplify	

### RandStream Objects: Generate random numbers using Threefry and Philox algorithms

When creating a random stream with RandStream, you now can use the Threefry and Philox random number generation algorithms.

### GraphPlot Object: Customize node and edge labels with font properties

GraphPlot objects have several new properties to enable customization of node and edge labels in plots of directed or undirected graphs.

Property	Description
NodeLabelColor	Background color of label
EdgeLabelColor	
NodeFontSize	Font size for label
EdgeFontSize	
NodeFontName	Font for label
EdgeFontName	
NodeFontAngle	Normal or italic text
EdgeFontAngle	

Property	Description
NodeFontWeight	Normal or bold text
EdgeFontWeight	
Interpreter	Interpretation of text characters in labels (none, tex, or latex)
ArrowPosition	Position of arrow on directed edges

## Version History

The new GraphPlot property `Interpreter` has a default value of `'tex'`. In previous releases, graph node and edge labels displayed text as the literal characters instead of interpreting the text using TeX markup. If you do not want node and edge labels to use TeX markup, then set the `Interpreter` property to `'none'`.

## **sinpi and cospi Functions: Compute the sine and cosine of multiples of $\pi$**

The `sinpi` and `cospi` functions compute the values of  $\sin(\pi x)$  and  $\cos(\pi x)$ . The answers provided by these functions are more accurate than answers provided by `sin(pi*x)` or `cos(pi*x)` because they do not compute `pi*x` explicitly. This convention compensates for roundoff error in the floating-point value of `pi`.

## Graphics

### Axes Interactions: Explore data with panning, zooming, data tips, and 3-D rotation enabled by default

Interactively explore your data using axes interactions that are enabled by default. For example, you can use the scroll-wheel to zoom into your data or hover over a data point to see a data tip. Also, you can click and drag the axes to pan the axes (2-D view) or rotate the axes (3-D view). For more information, see [Interactively Explore Plotted Data](#).

#### Version History

In previous releases, none of the interactions were enabled by default. To control if the axes interactions are enabled by default, use the `disableDefaultInteractivity` and `enableDefaultInteractivity` functions.

### Axes Toolbar: Access and customize a data exploration toolbar for each Axes object

Axes have a toolbar that appears above the top-right corner for quick access to the data exploration tools. The buttons available in the toolbar depend on the contents of the axes. The toolbar typically includes buttons to brush data, add data tips, rotate the axes (3-D axes only), pan or zoom the data, and restore the view.



You can customize the buttons available in the toolbar using the `axtoolbar` and `axtoolbarbtn` functions.

#### Version History

In previous releases, the buttons that now appear in the axes toolbar appeared in the figure toolbar instead. You can turn off the axes toolbar by setting the `Visible` property of the `AxesToolbar` object to `'off'`.

```
ax = gca;
ax.Toolbar.Visible = 'off';
```

You can restore the figure toolbar buttons using the `addToolbarExplorationButtons` command.

### Geographic Plots: Create line, scatter, and point density plots on interactive maps and control properties of a geographic axes

Create line, scatter, and point density plots on interactive maps and control properties of a geographic axes. Use the `geoplot`, `geoscatter`, and `geodensityplot` functions to create these plots. The `geolimits` function now works with any of these geographic plots, in addition to geographic bubble charts. To change the basemap used by any of these geographic plots or charts, use the new `geobasemap` function.

## stackedplot Function: Plot variables of a table or timetable for comparison using a common x-axis

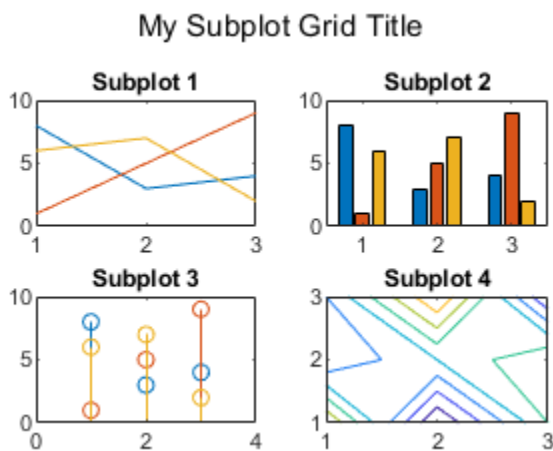
Plot the variables of a table or timetable. To ease visual comparison, the `stackedplot` function provides a common x-axis and separate y-axes for the variables.

## scatterhistogram Function: Visualize grouped data as a scatter plot with marginal histograms

To create a scatter plot with marginal histograms, use the `scatterhistogram` function. To group the data, you can use either the `'GroupVariable'` name-value pair argument with tabular data or the `'GroupData'` name-value pair argument with arrays.

## sgtitle Function: Create a title for a grid of subplots

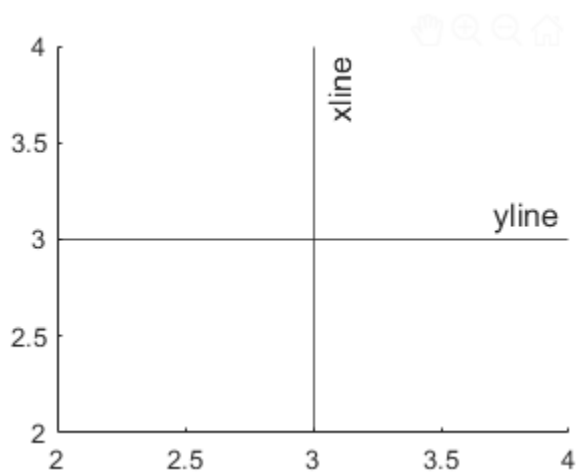
You can add an overall title to a grid of subplots, in addition to adding a title to each individual subplot. To add an overall title, use the `sgtitle` function.



## xline and yline Functions: Add vertical or horizontal lines to a plot

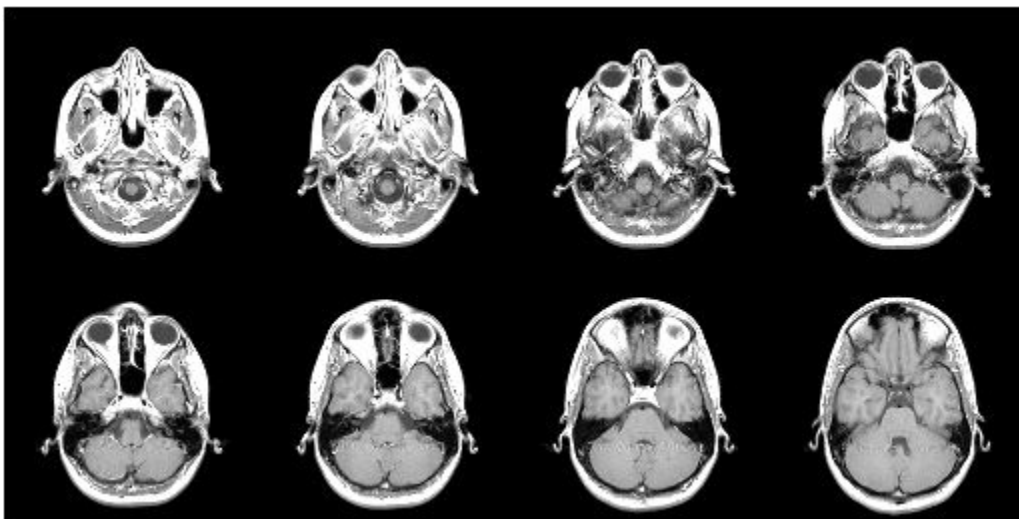
To add vertical or horizontal lines to a plot, use the `xline` or `yline` functions, respectively. For example, `xline(3)` plots a vertical line at  $x = 3$ .





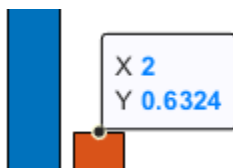
### **imtile Function: Combine multiple image frames into one rectangular tiled image**

To combine multiple image frames into one rectangular tiled image, use the `imtile` function.



### **Data Tips: Use TeX or LaTeX markup in data tips with improved visual appearance**

Data tips have an improved visual appearance with new text colors.



Also, data tips now display text characters using TeX markup by default. Control the interpretation of the text characters using the `Interpreter` property of the data cursor mode object. Set the property

value to `'tex'` for TeX markup (default), `'latex'` for LaTeX markup, or `'none'` for literal characters.

```
d = datacursormode;
d.Interpreter = 'latex';
```

## Version History

In previous releases, data tips displayed text as the literal characters instead of interpreting the text using TeX markup. If you do not want data tips to use TeX markup, then set the `Interpreter` property to `'none'`.

## Functionality being removed or changed

### legend function interprets argument as property name when property exists

*Behavior change*

Starting in R2018b, if you pass an argument to the `legend` function that matches the name of a legend property, the function interprets the argument as the name of a name-value pair. In previous releases, the `legend` function recognized name-value pairs only when the first argument was a cell array.

As a result of this change, in most cases, it is unnecessary to specify the first argument as a cell array when using name-value pairs. However, if you want a label in your legend that matches the name of a legend property, such as `Position` or `NumColumns`, then you *must* specify all the labels in a cell array. Otherwise, the `legend` function interprets the argument as a name-value pair instead of a label.

Description	Recommended Code
If you want a label in your legend that matches the name of a legend property, such as <code>'NumColumns'</code> , then specify all the labels in a cell array. If you specify <code>'NumColumns'</code> outside of a cell array, the <code>legend</code> function interprets it as a name-value pair.	<code>legend({'Label1', 'NumColumns', 'Label3', 'Label4'}, 'NumColumns', 'Value')</code>
If none of your labels match the name of a legend property, then you do not need to use a cell array around the labels.	<code>legend('Label1', 'Label2', 'Label2')</code>

### alpha and shading set both FaceColor and FaceAlpha properties

*Behavior change*

When updating surface and patch objects, the `alpha` and `shading` functions sometimes set both the `FaceColor` and `FaceAlpha` properties. These functions set both properties in cases where setting just one property results in a rendering issue. No updates to your code are required.

In previous releases, the `alpha` function set only the `FaceAlpha` property. Similarly, the `shading` function set only the `FaceColor` property.

## Data Import and Export

### Import Tool: Generate improved code when importing from spreadsheets

The **Import Tool** now offers improved code generation functionality for importing spreadsheets across platforms. For example, you can import datetimes on Mac and Linux and generate code that is easy to read. For more information, see [Read Spreadsheet Data Using Import Tool](#).

### Version History

For more information, see “Import Tool handling of spreadsheet dates and times and fields that are empty, unimportable, or error causing” on page 8-14.

### Web-Based Data: Read from web-based data sources like Amazon Web Services and Azure Blob Storage using `readtable`, `detectImportOptions`, `spreadsheetDatastore`, `imread`, and `imfinfo`

You can access tabular data and images from files stored in remote locations (Amazon S3, Windows Azure Blob Service, and HDFS) using these functions:

- `readtable`
- `detectImportOptions`
- `spreadsheetDatastore`
- `imread`
- `imfinfo`

When reading data from remote locations, you must specify the full path using a uniform resource locator (URL). For example, read a `csv` file from Amazon S3 cloud:

```
T = readtable('s3://bucketname/path_to_file/my_text_file.csv');
```

For more information on setting up MATLAB to access your online storage service, see [Work with Remote Data](#).

### write Function: Write tall arrays in a variety of formats to local or remote locations

The functionality of `write` is expanded to support additional formats and storage locations:

- Output formats include `.txt`, `.csv`, `.xls`, and more.
- Name-value pairs to control format-specific options, such as `WriteVariableNames` and `Delimiter`.
- Extended support for writing `.seq` and `.mat` formats to all supported file systems.
- Write data to remote locations in Amazon S3 or Windows Azure Blob Storage (WASBS).

## **stlread and stlwrite Functions: Read from and write to STL (Stereolithography) files for triangulations**

The `stlread` function enables you to read triangulation information from an STL file to create a triangulation object. You also can write a triangulation object or a 2-D `deLaunayTriangulation` object to a binary STL file with `stlwrite`.

## **TabularTextDatastore Object: Import data containing dates and times from non-English locales**

The `TabularTextDatastore` object now supports import of date and time data from non-English locales. For example, to create a datastore for reading files that contain German date and time data, set the `DatetimeLocale` parameter to `'de_DE'`.

```
ds = tabularTextDatastore('myDataFile.csv','DatetimeLocale','de_DE')
```

## **readtable and writetable Functions: Read or write spreadsheet files without initiating Microsoft Excel for Windows on Windows platforms**

On Windows platforms, you can choose not to open an instance of Microsoft Excel when reading or writing data from spreadsheet files. Set the `'UseExcel'` parameter to one of these values:

- `true` — Open an instance of Microsoft Excel to read (or write) the file. This setting is the default for Windows systems with Excel installed.
- `false` — Do not open an instance of Microsoft Excel to read (or write) the file. Using this setting might cause the data to be written differently for files with live updates like formula evaluation or plugins.

For more information, see `readtable` and `writetable`.

## **readtable Function: Manage the import of empty fields using import options**

You can manage empty fields in tabular data by using `readtable` along with import options. Use the `EmptyFieldRule` of import options object to specify how `readtable` handles empty fields. For more information, see `setvaropts`.

## **Scientific File Format Libraries: CFITSIO Library upgraded to version 3.420**

The CFITSIO library is upgraded to version 3.420.

## **Functionality being removed or changed**

### **Import Tool handling of spreadsheet dates and times and fields that are empty, unimportable, or error causing**

*Behavior change*

Starting in R2018b, the **Import Tool** app has improved functionality for importing data from spreadsheet files. The changes to imported data are minimal and are limited to uncommon cases:

- Empty or unimportable cells in spreadsheets:
  - The value you use to replace empty cells and unimportable cells must be the same. Previously, **Import Tool** allowed for different values for empty cells and unimportable cells.
  - Previously, **Import Tool** allowed you to specify a target string for cells that are unimportable. This feature is no longer supported.
- Date and time data in spreadsheets:
  - Import date and time data as MATLAB `datetime` arrays on all platforms.
  - Import numbers as date and times on all platforms.
  - Import Excel dates as numbers on all platforms.
- When running the **Import Tool** app in MATLAB on a Windows machine, cells in Excel spreadsheets with error conditions are no longer displayed.

### Basic parameter of the `readtable` function (Not Recommended)

*Still runs*

The `Basic` parameter of the `readtable` function is not recommended. Use the parameter name `UseExcel` instead. There are no plans to remove the `Basic` parameter at this time.

### UseExcel parameter of the `readtable` and `writetable` functions

*Behavior change in future release*

In future releases, the default value of the `UseExcel` parameter will be changed to `false`. The current default setting for `UseExcel` on Windows systems with Excel installed is `true`.

This table shows the typical usage of `readtable` and how to update your code to preserve the current behavior in future releases.

Code (R2018b and earlier)	Use this instead (Future releases)	Behavior
<code>T = readtable(filename)</code>	<code>T = readtable(filename, 'UseExcel', true)</code>	Starts an instance of Microsoft Excel when reading the file.

For more information, see `readtable` and `writetable`.

### Output from the `audioread` function for A-law or mu-law wave files

*Behavior change*

When reading in native mode, the `audioread` function returns data from A-law or mu-law wave files as `int16`.

Previously, `audioread` returned data from A-law or mu-law wave files as `int8`.

### Transparency output from the `imread` function

*Behavior change*

The `imread` function returns transparency information for indexed PNG images. For example, reading a png file returns a nonempty transparency array `alpha`:

```
[img,map,alpha] = imread('myIndexedImage.png');
whos alpha
```

Name	Size	Bytes	Class	Attributes
alpha	50x120	48000	double	

Previously, the `imread` function did not return transparency information for indexed PNG images. For example, previously reading a png file returned an empty transparency output `alpha`:

```
[img,map,alpha] = imread('myIndexedImage.png');  
whos alpha
```

Name	Size	Bytes	Class	Attributes
alpha	0x0	0	double	

## Data Analysis

### Vector Dimension Argument: Operate on multiple dimensions at a time for selected reduction functions

These functions now accept a vector dimension argument to specify multiple operating dimensions at a time, as well as the option 'all' to specify all dimensions of an array.

all	min
any	mode
bounds	prod
max	std
mean	sum
median	var

For example, `sum(A, 'all')` sums all the elements in a matrix A, and is equivalent to `sum(A, [1 2])`.

### grouptransform Function: Transform table or timetable data by groups

You can use the `grouptransform` function to perform group computations, such as normalization or filling missing data on table and timetable variables. For example, `g = grouptransform(T, 'School', 'norm')` normalizes the data in a table T by school using the vector 2-norm.

### groupsummary Function: Perform group summary computations on matrices

You now can group by matrix rows to perform summary computations using the `groupsummary` function.

### tall Arrays: Write custom algorithms to operate on tall arrays

The functions `matlab.tall.transform` and `matlab.tall.reduce` enable you to write custom algorithms to execute on tall arrays. These functions enable you to implement a range of parallelizable algorithms. `matlab.tall.transform` applies a single function to each block of a tall array, while `matlab.tall.reduce` is similar to MapReduce, where two functions are applied to a tall array, with the output of the first function being fed as input to the second function.

`matlab.tall.transform` and `matlab.tall.reduce` provide the flexibility to implement functions that otherwise do not currently support tall arrays. For more information, see [Develop Custom Tall Array Algorithms](#).

## tall Arrays: Operate on tall arrays with more functions, including conv2, wordcloud, and groupsummary

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the **Extended Capabilities** section at the bottom of the reference pages.

conv2	islocalmin
convn	rmoutliers
corrcoef	vecnorm
groupsummary	wordcloud
islocalmax	

In addition, some functions have expanded support for tall arrays. This expanded support removes some limitations of using these functions with tall arrays.

Functions	Added Support
join	Joining two tall inputs (tall tables and/or tall timetables) is now supported.  <i>Previously, one of the inputs was required to be an in-memory table or timetable.</i>
sort sortrows topkrows unique	These functions now support multiple output arguments. The extra outputs return indices that are relevant to the operation or that describe the location of interesting elements.  <i>Previously, these functions did not support multiple outputs for tall arrays.</i>
mean	Calculating the mean of datetime values is now supported.  <i>Previously, this function did not support tall arrays with an underlying data type of datetime.</i>

### rmoutliers Function: Remove outliers in an array, table, or timetable

The `rmoutliers` function detects and removes outlier data in an array, table, or timetable.

### islocalmin and islocalmax Functions: Specify a range of data for prominence computation

The 'ProminenceWindow' name-value pair for the `islocalmin` and `islocalmax` functions enables you to specify a local neighborhood about each element in the input data when computing the corresponding prominence.



## Table and Timetable Metadata: Store custom metadata for each variable

You can store custom metadata for variables of a table or timetable using its `Properties.CustomProperties` object. For more information, see the Custom Metadata sections of `table` and `timetable`.

## timetable Data Type: Save memory when storing row times with regular time steps

If you create a timetable with regular time steps, then it stores the row times using less memory than in previous releases. You can create a regular timetable using the `array2timetable`, `retime`, `synchronize`, `table2timetable`, or `timetable` functions.

A timetable stores the start time, time step, and sample rate as properties. For more information, see the Row Times Metadata section of `timetable`.

## timerange Function: Specify unit of time to define time range

When you specify the beginning and end of a time range using datetime values, you also can specify the date or time component used to define the endpoints. For example, `S = timerange('2018-9-30', 'quarters')` creates a time range spanning all dates in the third quarter of 2018, since September 30 is in the third quarter. For more information, see `timerange`.

## convertvars Function: Convert table or timetable variables to specified data type

You can specify variables of a table or timetable, and convert them to a different data type, using the `convertvars` function.

## table, timetable, and addvars Functions: Use single quotes for input names, not double-quoted strings

When creating or updating a table or timetable using the `table`, `timetable`, or `addvars` functions, use single quotes for input names (such as `'Size'`, `'VariableNames'`, `'After'`, or `'TimeStep'`) to avoid confusion with variable inputs. Variables and input values can use double-quoted strings. For example:

```
T = table("some text",'VariableNames',["X"]);
```

## Functionality Being Removed or Changed

### 'SamplingRate' is not recommended

*Still runs*

The `'SamplingRate'` name-value pair argument is not recommended. Use `'SampleRate'` instead. The corresponding timetable property is also named `SampleRate`.

For backward compatibility, you still can specify `'SamplingRate'` as the name of the name-value pair. However, the value is assigned to the `SampleRate` property.

This change in behavior affects the timetable functions shown in the table.

Function
<code>array2timetable</code>
<code>retime</code>
<code>synchronize</code>
<code>table2timetable</code>
<code>timetable</code>

### **Default random number generator change for `tallrng`**

*Behavior change in future release*

Starting in R2019a, the default random number generator for `tallrng` will change to `threefry`. This generator offers performance enhancements for parallel calculations over the current default. In releases up to R2018b, the default random number generator for `tallrng` is `combRecursive`.

With a different default generator, MATLAB will generate different sequences of random numbers by default in the context of tall arrays. However, the statistics of these calculations will remain unaffected. Therefore, you should update any code that relies on the specific random numbers being generated. However, most calculations on the random numbers should be unaffected.

To set the generator to the settings used by default in R2018b and earlier releases, use the command:

```
tallrng(0, 'combRecursive')
```

## App Building

### **App Designer: Add and configure date selection components on the App Designer canvas**

Drag and drop date picker components from the **Component Library** onto the canvas.

### **App Designer: Unified property inspector in Design View and Code View**

The **Component Properties** pane in Design View now provides the full list of properties and the same interactive controls as Code View does.

### **App Designer: Expand and collapse sections of code in Code View**

App Designer provides code folding capabilities in Code View. You can expand sections of code that you are working on and collapse other sections to navigate your code more easily.

### **App Designer: Export apps as code files**

Export apps you create in App Designer as (.m) program files. Select **Save > Export to .m File** in the App Designer toolstrip.

Exporting an app as a program file enables you to change it outside of App Designer. However, there is no option for importing your changes back into App Designer.

### **App Designer: Locate errors and warnings in your code with the Code Analyzer message bar**

App Designer now provides the same Code Analyzer messaging system as the MATLAB Editor does.

### **App Designer: Program apps faster using improved code suggestions and completions**

As you code your app, App Designer displays the same contextual hints for arguments, property values, and syntaxes as the Live Editor does.

### **App Designer: Control App Designer Code View settings using MATLAB preferences**

Control the following settings for App Designer Code View by setting MATLAB preferences:

- Highlight current line
- Show line number
- MATLAB syntax highlighting colors

When you set any of these preferences, the change applies to the MATLAB Editor and App Designer.

## uigridlayout Function: Configure app layouts using a grid layout manager

Use the `uigridlayout` function to create grid layout managers in your apps. When you use a grid layout manager, you position UI components along the rows and columns of an invisible grid that spans the entire app window or a container in the window. Using rows and columns to position components is easier to manage than setting pixel values in `Position` vectors. For resizable apps, grid layouts provide more flexibility than the automatic resize behavior in App Designer, and they are easier to code than `SizeChangedFcn` callbacks. For more information, see [Using Grid Layout Managers](#).

Grid layout managers are only available in apps created with the `uifigure` function.

## Scrolling Containers: Enable scrolling for figure, panel, tab, and button group containers

Enable scrolling within a figure or any child container by setting the `Scrollable` property to `'on'`. To scroll to a location within the container programmatically, call the `scroll` function.

Scrolling works only in App Designer apps, in figures created with the `uifigure` function, or in child containers within those figures.

## Figure Interactions: Create apps with custom mouse and keyboard interactions using figures created with the uifigure function

Create apps that respond to custom mouse and keyboard interactions. The following properties are available in App Designer and in figures created with the `uifigure` function.

Category	Properties
Figure Data	<code>SelectionType</code> , <code>CurrentPoint</code> , <code>CurrentCharacter</code>
Mouse Callbacks	<code>ButtonDownFcn</code> , <code>WindowButtonDownFcn</code> , <code>WindowButtonUpFcn</code> , <code>WindowButtonMotionFcn</code> , <code>WindowScrollWheelFcn</code>
Keyboard Callbacks	<code>KeyPressFcn</code> , <code>KeyReleaseFcn</code> , <code>WindowKeyPressFcn</code> , <code>WindowKeyReleaseFcn</code>

Figures created with the `figure` function continue to support these properties as they have in the past.

## Graphics Support: Integrate plots into an app using the axes, polaraxes, and geoaxes functions

Display more types of plots in App Designer apps and in figures created with the `uifigure` function. The expanded list of plots includes subplots, polar plots, and geographic plots. For more information, see [Displaying Graphics in App Designer](#).

## Tooltips: Create custom tooltips for UI components in apps

Set the `Tooltip` property on a UI component to display a tooltip when the user hovers the mouse over the component at run time. Tooltips display even when the components are disabled.

The `Tooltip` property is available for UI components in App Designer apps and in figures created with the `uifigure` function.

If you are creating an app using `GUIDE` or the `figure` function, use the `Tooltip` property instead of the `TooltipString` property on `UIControl`, `Table`, `Tab`, `PushTool`, and `ToggleTool` UI components. For details, see “`TooltipString` property is not recommended” on page 8-23.

## Deployed Web Apps: Access files in deployed web apps using the `uigetfile` and `uiputfile` functions

Call `uigetfile` in a web app to enable users to select files on their local systems. Users can select files by dropping them onto the app or by selecting them in a file browser.

Call `uiputfile` to enable users to specify file names for saving files.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

## Running Apps in Browsers: Use most modern browsers to run apps in MATLAB Online or as deployed web apps

Run apps in MATLAB Online, and run deployed web apps using Safari, Firefox, and Microsoft Edge in addition to Google Chrome and Internet Explorer. Use the current versions of these browsers for best results. For the best overall experience, use Google Chrome.

For information about MATLAB Online system requirements, see <https://www.mathworks.com/support/requirements/browser-requirements.html>.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

## `uisetcolor` Function: Select custom colors interactively

Select custom colors using an improved interactive color picker in the `uisetcolor` dialog box. Make your selection by clicking within a color gradient. You can select colors as HSV values in addition to the RGB and hexadecimal options that have been available in previous releases.

## Functionality Being Removed or Changed

### **TooltipString property is not recommended**

*Still runs*

The `TooltipString` property for the `UIControl`, `Table`, `Tab`, `PushTool`, and `ToggleTool` UI component objects is not recommended. Use the new `Tooltip` property to display a tooltip instead.

The `TooltipString` property is no longer listed when you call the `get`, `set`, or `properties` functions to list the properties of the object. There is no plan at this time to remove support for getting or setting the value of the `TooltipString` property. However, partially specifying the `TooltipString` property name might produce errors. To prevent the error, specify the full name of the `TooltipString` property, or use the `Tooltip` property. Both property names correspond to the same value.

## Performance

### **Startup: Increased speed of MATLAB startup**

MATLAB starts faster because of continued infrastructure improvements and optimizations.

### **Execution Engine: Index into large arrays with improved performance when using the colon operator**

Now colon indexing into large, numeric arrays is faster.

### **Execution Engine: Faster calls to built-in functions**

Calls to built-in functions are faster due to reduced overhead.

### **Live Editor: Create new and open existing live scripts faster**

New and existing live scripts open faster than in previous releases.

### **Enumerations: Improved set function performance with enumerations**

When called with enumeration arrays, execution of set operation functions such as `ismember` is faster.

### **Building Apps: Faster canvas interactions in App Designer**

Many common tasks in the App Designer canvas, such as copying, pasting, and deleting components are 90% faster.

### **Running Apps: Faster startup time for apps**

Startup time is 10% to 30% faster for apps created in App Designer and apps created programmatically using the `ui figure` function. The time savings become more noticeable as the number of components in your app increases.

### **sort Function: Sort matrices and arrays faster**

The `sort` function is now faster when sorting numeric matrices or multidimensional arrays.

## Hardware Support

### MATLAB Online: Communicate with Raspberry Pi hardware board from MATLAB Online

You now can connect to and control Raspberry Pi hardware boards remotely from MATLAB Online. Raspberry Pi 2 Model B and Raspberry Pi 3 Model B are supported. Install the MATLAB package onto Raspberry Pi, use `raspi` in MATLAB Online to discover available boards, and use `raspi` to create a connection. The functions in the MATLAB Support Package for Raspberry Pi Hardware (except `openShell` and `putFile`) are available in MATLAB Online. For more information on how to communicate with your Raspberry Pi in MATLAB Online, see [Connect to Raspberry Pi Hardware Board in MATLAB Online \(MATLAB Support Package for Raspberry Pi Hardware\)](#).

### Deploy a MATLAB function on Raspberry Pi hardware

From R2018b, MATLAB Support Package for Raspberry Pi Hardware enables you to deploy your MATLAB function as a standalone executable on the Raspberry Pi hardware. For deploying the function, use the `targetHardware` command to create a Raspberry Pi configuration object, and then use the `deploy` command to deploy the function on the Raspberry Pi hardware.

To use this feature, you must install MATLAB Coder in your computer.

To support deployment, the Raspberry Pi functions are enhanced to generate code. Some functions are listed here. For more details, see [MATLAB Support Package for Raspberry Pi Hardware](#).

Peripheral	Function
Raspi	<code>raspi</code>
LEDs	<code>writeLED</code>
GPIO Pins	<code>configurePin</code> , <code>readDigitalPin</code> , and <code>writeDigitalPin</code>
I2C Interface	<code>i2cdev</code> , <code>read</code> , <code>write</code> , <code>readRegister</code> , and <code>writeRegister</code>
SPI Interface	<code>spidev</code> and <code>writeRead</code>
Serial Port	<code>serialdev</code> , <code>read</code> , and <code>write</code>
Servo	<code>servo</code> and <code>writePosition</code>
Linux	<code>system</code>
Camera Board	<code>cameraboard</code> , <code>snapshot</code> , <code>record</code> , and <code>stop</code>
Web Camera	<code>webcam</code> and <code>snapshot</code>
Pulse Width Modulation	<code>writePWMFrequency</code> , <code>writePWMDutyCycle</code> , and <code>writePWMPulse</code>

### iOS and Android Sensors: Acquire sensor data when your device does not have network access

You can acquire sensor data locally on your Android™ or Apple iOS device, with or without a network connection. This method is an alternative method of collecting the sensor data instead of streaming it

from the device to your computer running MATLAB. It is especially useful if you want to collect sensor data while your device does not have a network connection.

To use this method of acquiring sensor data, you log sensor data locally on your mobile device using MATLAB Mobile, and then upload the files to MATLAB Drive once you are connected. To use MATLAB Drive, you must log into your MathWorks account. Alternatively, you can transfer the log files manually using a USB cable. Once you have the sensor data on your computer running MATLAB, you use the MATLAB Support Package for Android Sensors or the MATLAB Support Package for Apple iOS Sensors to view and analyze the data.

For more information, see the help screens in MATLAB Mobile or the MATLAB Support Package for Android Sensors or MATLAB Support Package for Apple iOS Sensors documentation.

## **iOS and Android Sensors: Upload sensor logs from the device to MATLAB Drive**

You can acquire sensor data locally on your Android or Apple iOS device, with or without a network connection. You can then upload the files to MATLAB Drive when you are connected. To use MATLAB Drive, you must log into your MathWorks account.

You can have the log files automatically upload to MATLAB Drive when you have a network connection, or choose to upload them from the **Sensor Logs** screen in MATLAB Mobile any time. Go to **Settings > Configure > MATLAB Drive Upload > Auto Upload** to select your preference.

For more information, see the help screens in MATLAB Mobile or the MATLAB Support Package for Android Sensors or MATLAB Support Package for Apple iOS Sensors documentation.



## Advanced Software Development

### **Tab Completion: Validate function signature file with validateFunctionSignaturesJSON function**

Use the `validateFunctionSignaturesJSON` function to validate the JSON-formatted file that contains information about your function signatures. For more information, see `validateFunctionSignaturesJSON`.

MATLAB uses the information in `functionSignatures.json` to improve interactive features, such as tab completion and function hints. For information on creating a `functionSignatures.json` file, see [Customize Code Suggestions and Completions](#).

### **Tab Completion: JSON parser for functionSignatures.json upgrade**

The JSON file parser that MATLAB uses to read `functionSignatures.json` files is upgraded for R2018b.

### **Version History**

The updated parser has stricter validation for JSON files. Before R2018b, a `functionSignatures.json` file could have syntax errors that were undetected by the JSON parser. For these newly detected errors, MATLAB displays an error message in the Command Window when it reads the file.

Correct any syntax errors in the JSON file. Best practice is to validate `functionSignatures.json` files with the `validateFunctionSignaturesJSON` function.

### **Java SE 8: MATLAB support, providing improved security and access to new Java features**

Java interface supports JRE version Java 1.8.0\_152. For more information, see [MATLAB Supported Interfaces to Other Languages](#).

### **Python Interface: Pass multidimensional numeric or logical arrays between MATLAB and Python**

MATLAB auto converts numeric and logical array data input to a Python function to a Python `memoryview` object. For more information, see [Passing Matrices and Multidimensional Arrays](#).

### **C++ MEX API: Call MATLAB asynchronously from within a MEX file using the C++ API**

Use the asynchronous C++ MEX API to call MATLAB functions from user application threads in MEX functions. Calls to MATLAB from user application threads are queued and executed in sequence with other MATLAB commands. For more information, see [Call MATLAB from Separate Threads in MEX Function](#).

## Unit Testing Framework: Run tests in parallel with more plugins and more intelligent scheduling

R2018b includes refinements to the testing framework parallel scheduling algorithm. These enhancements improve the overall performance of the 'UseParallel' option in `runtests` and the `TestRunner.runInParallel` method.

Also, the following plugins now can run tests in parallel:

- `CodeCoveragePlugin` with Cobertura format
- `TestReportPlugin`
- `XMLPlugin.producingJUnitFormat`

## Unit Testing Framework: Use external parameters in parameterized test

You can inject variable inputs into your parameterized test. For example, you can specify that a test uses input data from a file instead of the data hard-coded within a test. To define external parameters, use the `fromData` method of the `matlab.unittest.parameters.Parameter` class. Then, specify that your parameterized test use the external parameters using the 'ExternalParameters' option to `TestSuite` creation methods. `TestSuite` creation methods include `fromClass`, `fromFile`, `fromFolder`, `fromMethod`, `fromName`, and `fromPackage`.

For more information, see [Use External Parameters in Parameterized Test](#).

## Unit Testing Framework: Sort test suite based on shared fixtures

To reduce shared fixture setup and teardown operations, sort test suite elements so that elements that require the same shared fixture setup are adjacent. To sort an existing test suite, use the `sortByFixtures` method of `matlab.unittest.TestSuite`. The `testsuite` function automatically creates a test suite that is sorted based on shared fixtures. However, if you concatenate test suites after creating them, call the `sortByFixtures` method to reorder the suite. For more information, see `matlab.unittest.TestSuite.sortByFixtures`.

## Unit Testing Framework: Explicitly control output display detail and logged diagnostic level

From the Run Tests section in the Editor, you can control the amount of detail displayed for a test run. For example, to display the most information, select **Verbose** from the Output Detail test option under the **Run Tests** icon. To suppress output, select **None**.

You can control which logged diagnostics are displayed by selecting a value from the Logging Level test option under the **Run Tests** icon. Logged diagnostics are diagnostics that you supply in your test code with a call to the `TestCase.log` method. MATLAB reports logged diagnostics at the specified logging level and lower. For example, to exclude diagnostics logged at detailed or verbose levels, select **Concise**.

You can also control the output detail and logged diagnostic level programmatically using the 'OutputDetail' and 'LoggingLevel' name-value pairs in these features:

- `runtests` function
- `TestRunner.withTextOutput` method
- `DiagnosticsOutputPlugin` class
- `DiagnosticsRecordingPlugin` class
- `TAPPlugin` class
- `XMLPlugin` class ('OutputDetail' only)
- `TestReportPlugin` class ('LoggingLevel' only)

The `matlab.unittest.Verbosity` enumeration now contains the `Verbosity.None` member. Use this verbosity level to indicate a detail level that includes no information. This enumeration member is accepted anywhere that accepts a `Verbosity` value, except for the `matlab.unittest.TestCase.log` and `matlab.unittest.fixtures.Fixture.log` methods. These methods direct the framework to log diagnostics, not to display them.

## Version History

Before R2018b, the 'Verbosity' name-value pair controlled both the output detail and the logged diagnostic level, and the 'ExcludingLoggedDiagnostics' name-value pair determined whether plugins recorded logged diagnostics.

These name-value pairs are supported, but are not recommended. Use 'LoggingLevel' and 'OutputDetail' instead. Replace instances of 'ExcludingLoggedDiagnostics' and 'Verbosity' in the following plugins:

- `runtests` function ('Verbosity' only)
- `matlab.unittest.TestRunner.withTextOutput` method ('Verbosity' only)
- `matlab.unittest.plugins.DiagnosticsRecordingPlugin` class
- `matlab.unittest.plugins.TAPPlugin` class
- `matlab.unittest.plugins.TestReportPlugin` class

## Unit Testing Framework: Configure detail level of output diagnostics

To configure the amount of detail included in an output stream for diagnostics from passing, failing, and logging events, add the `DiagnosticsOutputPlugin` to a `TestRunner` instance. For example, you can specify that Command Window output includes passing diagnostics at a verbose detail level or you can suppress the display of diagnostics.

For more information, see the `matlab.unittest.plugins.DiagnosticsOutputPlugin` class.

## Unit Testing Framework: Compare values faster when using constraints

The `matlab.unittest.constraints.IsEqualTo` class has improved performance when comparing equal values.

## Version History

Before R2018b, the `IsEqualTo` constraint called `isequal` or `isequaln` to determine if the actual and expected objects were equal. With release R2018b, the constraint can, sometimes, determine

that the actual and expected objects are equal without calling these functions. In these cases, if the objects being compared overload the `isequal` or `isequaln` functions, then whatever specialized behavior these methods define is not used in the comparison.

## **App Testing Framework: Programmatically choose tree node**

The `choose` method now supports programmatic selection of tree nodes in your app tests. For more information, see `matlab.uitest.TestCase.choose`.

## **Performance Testing Framework: Measure execution time of fast code more accurately with the `TestCase.keepMeasuring` method**

Performance tests that execute too quickly for MATLAB to time accurately are filtered with an assumption failure. With the `keepMeasuring` method, the testing framework can measure significantly faster code by automatically determining the number of times to iterate through code and measuring the average performance. Use the `keepMeasuring` method within the condition of a `while` loop. For more information, see `matlab.perftest.TestCase.keepMeasuring` and `Measure Fast Executing Test Code`.

## **Mocking Framework: Invoke function upon mocked method call**

You can specify that, each time you call a mocked method, it calls another function. For example, specify that each time you call a mocked `roll` method, it calls the `randi` function.

To specify that a mock method uses a function handle to invoke another function, define behavior with the `Invoke` class of the `matlab.mock.actions` package. This action differs from the `AssignOutputs` action, which returns values that are defined when you create the `AssignOutputs` instance.

For more information, see `matlab.mock.actions.Invoke`.

## **Mocking Framework: Verify interactions on mock occurred in order**

You can create a constraint that is satisfied if interactions with a mock occurred in a specific order. Use the `'RespectingOrder'` option with `matlab.mock.constraints.Occurred` to verify that mock methods were called and properties were accessed and set in a particular order.

For more information, see `matlab.mock.constraints.Occurred`.

## **Mocking Framework: Clear history of recorded mock object interactions**

Use the `clearMockHistory` method of `matlab.mock.TestCase` to clear the history of recorded mock object interactions. For more information, see `matlab.mock.TestCase.clearMockHistory`.

## **`matlab.test.behavior.Missing` class: Verify class satisfies missing-value behavior contract**

Create a test class that derives from the `matlab.test.behavior.Missing` class to test if the missing value for the class satisfies the missing-value contract in MATLAB. If your class represents a

data type and you want MATLAB to treat missing values of your class similar to built-in classes, ensure that your class satisfies the missing-value contract.

Typically, you use the behavior test as part of a test-driven development workflow. If you want the missing value for your class to satisfy the missing contract with MATLAB, write the behavior test and modify the class under test until the test results are as you expect.

For more information, see `matlab.test.behavior.Missing`.

## MEX Functions: Build Fortran MEX Files with Interleaved Complex API

The Fortran Matrix API supports the interleaved storage representation of complex numbers. For more information, see MATLAB Support for Interleaved Complex API in MEX Functions.

---

**Note** To run a Fortran MEX file built with the interleaved complex API in MATLAB R2018a, you must use MATLAB R2018a Update 3.

---

### Version History

If you build Fortran MEX functions, then you should review the Do I Need to Upgrade My MEX Files to Use Interleaved Complex API? topic.

The functionality for several Fortran Matrix API functions has changed. For information, see:

- “Fortran Matrix API functions `mxGetPi`, `mxSetPi`, `mxGetImagData`, and `mxSetImagData` incompatible with interleaved complex API” on page 8-33
- “Change of behavior for Fortran Matrix API functions `mxGetPr`, `mxSetPr`, `mxGetData`, and `mxSetData`” on page 8-33
- “Change of behavior for Fortran Matrix API function `mxGetElementSize`” on page 8-33
- “Change of behavior for Fortran Matrix API functions `mxCopyComplex16ToPtr`, `mxCopyPtrToComplex16`, `mxCopyComplex8ToPtr`, and `mxCopyPtrToComplex8`” on page 8-33

### Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	MinGW-w64 version 6.3.0 compiler from <a href="https://mingw-w64.org">https://mingw-w64.org</a>	Windows
Added	Intel Parallel Studio XE 2018 with Microsoft Visual Studio 2015 and 2017 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2018 for Fortran	macOS
Discontinued	Microsoft Visual C++ <sup>®</sup> 2013 Professional	Windows

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

## System objects: Flexible requirements for inputs when calling System objects

In MATLAB, you can now call System objects with fewer inputs than those defined in the `stepImpl` or `outputImpl` methods. When the System object runs, the algorithm determines how inputs are used. This flexibility matches default MATLAB behavior for other functions and objects.

The System object algorithm might not be flexible for inputs depending on how the algorithm is implemented or if the System object implements the `getNumInputsImpl` method.

## System object authoring: Use enumerations to define finite property lists in System objects

When adding a System object property with a finite list of values, use enumerations to define the allowed values. To add enumerations, open your System object file in the MATLAB editor, and select **Insert Property > Enumerations**.

For information about converting StringSets to enumerations, see “System object authoring StringSet class will be removed” on page 8-34.

## Reference Architecture: Deploy and run MATLAB on Amazon Web Services (AWS) and Microsoft Azure

You can deploy and run the MATLAB desktop on AWS® and Azure® and connect to it using the Remote Desktop Protocol (RDP). Use a MathWorks provided AWS CloudFormation template to deploy to AWS and an Azure Resource Manager (ARM) template to deploy to Microsoft Azure. For more information, see MATLAB in the Cloud (<https://www.mathworks.com/cloud.html>). For reference architecture deployment details, see:

- MATLAB on AWS
- MATLAB on Azure

## Git Stashes: Store uncommitted changes for later use

Starting in R2018b, you can use Git stashes directly from MATLAB. You can create a Git stash to store uncommitted changes for later use. For details, see Use Git Stashes.

## Functionality being removed or changed

### **matlab.unittest.plugins.FailureDiagnosticsPlugin is not recommended**

*Still runs*

The `matlab.unittest.plugins.FailureDiagnosticsPlugin` class is not recommended. Use the `matlab.unittest.plugins.DiagnosticsOutputPlugin` class instead. There are no plans to remove `FailureDiagnosticsPlugin` at this time.

### **Programmatic dependence on specific diagnostic subclass from `getDiagnosticFor` method of constraint and tolerances**

Rework code that relies on properties or methods specific to `matlab.unittest.diagnostic.ConstraintDiagnostic` instances returned from the `getDiagnosticFor` method of `matlab.unittest.constraints` classes.

As of R2018b, diagnostics returned from constraint and tolerance classes in the `matlab.unittest.constraints` package are instances of `matlab.unittest.diagnostics.FrameworkDiagnostic`.

### **Protected access for `matlab.unittest.fixtures.Fixture.onFailure`**

*Behavior change in future release*

The `matlab.unittest.fixtures.Fixture.onFailure` method will have protected access in a future release. Currently, the `Fixture.onFailure` method has public access. However, this method is designed to be used in subclasses of the `Fixture` class. In a future release, the `Fixture.onFailure` method will have protected access, which restricts use of the method to subclasses of the `matlab.unittest.fixtures.Fixture` class.

### **Fortran Matrix API functions `mxGetPi`, `mxSetPi`, `mxGetImagData`, and `mxSetImagData` incompatible with interleaved complex API**

*Still runs*

Do not use `mxGetPi` and `mxSetPi` functions in Fortran MEX files built with the interleaved complex API (mex option `-R2018a`). Use `mxGetComplexDoubles` (Fortran) instead of `mxGetPr` and `mxGetPi`.

Do not use `mxGetImagData` and `mxSetImagData` functions in Fortran MEX files built with the interleaved complex API (mex option `-R2018a`). Use Typed Data Access functions instead.

For more information, see MATLAB Support for Interleaved Complex API in MEX Functions.

### **Change of behavior for Fortran Matrix API functions `mxGetPr`, `mxSetPr`, `mxGetData`, and `mxSetData`**

*Still runs*

Do not use the `mxGetPr` and `mxSetPr` functions in Fortran MEX files built with the interleaved complex API (mex option `-R2018a`) for complex arrays. Use these functions for real arrays only, or use Typed Data Access functions.

Do not use the `mxGetData` and `mxSetData` functions in Fortran MEX files built with the interleaved complex API (mex option `-R2018a`) for numeric arrays. Use these functions for nonnumeric arrays only. For numeric arrays, use Typed Data Access functions.

### **Change of behavior for Fortran Matrix API function `mxGetElementSize`**

*Behavior change in future release*

For a complex Fortran `mxArray` built with the interleaved complex API (mex option `-R2018a`), `mxGetElementSize` (Fortran) returns twice the value that the function in the separate complex API returns.

### **Change of behavior for Fortran Matrix API functions `mxCopyComplex16ToPtr`, `mxCopyPtrToComplex16`, `mxCopyComplex8ToPtr`, and `mxCopyPtrToComplex8`**

*Behavior change in future release*

The function signatures for Fortran Matrix API functions `mxCopyComplex16ToPtr` (Fortran), `mxCopyPtrToComplex16` (Fortran), `mxCopyComplex8ToPtr` (Fortran), and

`mxCopyPtrToComplex8` (Fortran) are different for MEX files built with the interleaved complex API (mex option `-R2018a`). The functions built with the separate complex API have two arguments for the data, the real and complex parts. The functions built with the interleaved complex API have a single argument for the data.

### System object authoring `StringSet` class will be removed

*Still runs*

The class `matlab.system.StringSet` will be removed in a future release. To bring System object infrastructure closer to MATLAB classes, regular MATLAB enumerations replace the System object-specific `StringSet` functionality. To define a finite set of properties in System objects, use enumerations instead.

### Update Code

StringSet Properties	Enumeration Properties
<pre>properties     Flavor = 'Chocolate' end  properties (Hidden,Constant)     FlavorSet = matlab.system.StringSet...         ({'Vanilla','Chocolate'}) end</pre>	<p>In your System object class, define the property:</p> <pre>properties     Flavor (1,1) FlavorValues end</pre> <p>In a separate file, define the enumeration class:</p> <pre>classdef FlavorValues &lt; int32     enumeration         Chocolate (0)         Vanilla (1)     end end</pre>

In the MATLAB Editor, use the **Insert Property > Enumeration** for help in creating the enumeration class.

## Version History

MATLAB enumerations do not support special characters or spaces for the enumeration values. When converting from `StringSets` to enumerations, remove or replace these characters in the enumeration class.

### `matlab.editor.autoformat.DoubleHashtagForHeading` setting has been removed

*Errors*

The `matlab.editor.autoformat.DoubleHashtagForHeading` setting has been removed. Use the `matlab.editor.autoformat.HashtagsForHeading` setting instead.

To update your code, change instances of the setting `matlab.editor.autoformat.DoubleHashtagForHeading` to `matlab.editor.autoformat.HashtagsForHeading`. For more information, see `matlab.editor.Settings`.



# R2018a

---

**Version: 9.4**

**New Features**

**Bug Fixes**

**Version History**



## Desktop

### Live Editor: Create live functions with richly formatted documentation, including equations and images

In the Live Editor, you can create live functions that accept inputs and return outputs. To document your live functions, add richly formatted text, which includes equations, images, and formatted code examples. Then, you can use the `doc` command to view the documentation in the Help browser. For more information, see [Create Live Functions](#).

### Live Editor: Debug live functions and scripts

To diagnose problems in live functions and live scripts, debug your code in the Live Editor. You can use several methods to debug in the Live Editor:

- Show output by removing semicolons.
- Run to a specific line of code and pause using the  button.
- Investigate called functions and scripts by stepping in using the  button.
- Add breakpoints to your file to enable pausing at specific lines when you run the file.

For more information, see [Debug Code in the Live Editor](#).


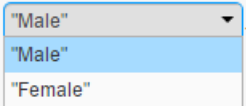
### Live Editor: Add sliders and drop-down lists to control variable values in a live script

You can add sliders and drop-down lists to your live scripts to interactively control variable values. Adding interactive controls to a live script is useful when you want to share the live script with others. Use the controls to define and limit the values of variables that others can change in your live script.

#### Determine Average Weight


Determine the average weight of a group of patients for a specified gender over a specified height.

```
load patients

selectedHeight = 68  ;
selectedGender =  ;

averageWeight = mean(Weight(Gender==selectedGender & Height>=selectedHeight))

averageWeight = 180.2750
```

To add a numeric slider or drop-down list, go to the **Insert** tab, click  **Control**, and select from the available options. For more information, see [Add Interactive Controls to a Live Script](#).

## Live Editor: Sort table data interactively

In the Live Editor, you can sort table data interactively. To sort data in a table, click the down arrow ▼ to the right of a variable name in the table and select from the available sorting options.

	LastName	Age	Gender
1	'Adams'		'Female'
2	'Alexander'		'Male'
3	'Allen'	39	'Female'
4	'Anderson'	45	'Female'
5	'Bailey'	38	'Female'
6	'Baker'	44	'Male'
7	'Barnes'	42	'Male'
8	'Bell'	45	'Male'
9	'Bennett'	35	'Female'
10	'Brooks'	39	'Male'

Use the **Update Code** button below the table to add the generated code to your live script. Adding the generated code to your live script ensures that the sorting is reproduced the next time you run the live script.

## Live Editor: Create a table of contents and add formatted code examples

You can create tables of contents in live scripts and functions that contain a list of all the titles and headings in the document. To insert a table of contents, go to the **Insert** tab and select **Table of Contents**. Only the title of the table of contents is editable.

You also can add formatted code examples to live scripts and functions. A code example is sample code that appears as text. To add a code example, go to the **Insert** tab, click **Code Example** and select **Plain**. The Live Editor displays the sample code as indented and monospaced text. To add a MATLAB syntax highlighted code example, go to the **Insert** tab, click **Code Example** and select **MATLAB**.

For more information about adding formatted text to live scripts and functions, see [Format Files in the Live Editor](#).

## Live Editor: Select and edit a rectangular area of code

In the Live Editor, you can select a rectangular area in your code (also known as column selection or block edit) by pressing the **Alt** key while making a selection. Selecting and editing a rectangular area of code is useful if you want to copy or delete several columns of data, or if you want to edit multiple lines at one time. For example, select the second column of data in A.

```
A = [ 10 20 30 40 50; ...
      60 70 80 90 100; ...
      110 120 130 140 150];
```

Type 0 to set all the selected values to 0.

```
A = [ 10 0 30 40 50; ...  
     60 0 80 90 100; ...  
     110 0 130 140 150];
```

## Add-Ons Explorer: Browse by category to discover convenient, helpful add-ons

Use the available categories to browse for new, convenient, and helpful add-ons in the Add-Ons Explorer. For more information about add-ons and how to find and install them, see [Get Add-Ons](#).

## Comparison Tool: Find differences in live scripts and functions

Use the Comparison tool to find differences in live scripts and functions. The Comparison tool highlights differences in both the code and the text.

To start a comparison of a live script or function, go to the **Live Editor** or **Home** tab, click **Compare**, and then select the files you want to compare. To start a comparison from the Current Folder browser, select a file, right-click, and select **Compare Against**.

## Favorites: Rerun favorite commands

Create favorite commands (previously called command shortcuts) to easily rerun a group of MATLAB language statements that you use regularly. To access existing favorite commands, go to the **Home** tab and in the **Code** section, click **Favorites** and then select from available favorite commands. To create a new favorite command, click **Favorites** and then press the **New Favorite** button. For more information, see [Rerun Favorite Commands](#).

## Toolbox Packaging: Specify portability information for custom toolboxes

You can indicate which platforms and MATLAB releases support your custom toolbox. When someone installs your toolbox on an unsupported platform or MATLAB release, MATLAB displays a warning. However, they can still install the toolbox. For more information, see [Create and Share Toolboxes](#).

## Language and Programming

### Empty Arrays: Create complex empty arrays using functions such as zeros and ones

You can create complex empty arrays using the `complex` function and functions that support complex input with the 'like' syntax, such as `zeros` and `ones`. For example, the commands `zeros(0,0,'like',1i)` and `complex([])` now return complex empty output instead of real empty output.

### Version History

Previously, the `complex` function and functions that support complex input with the 'like' syntax always returned real empty arrays when one or more dimensions were 0. To preserve this behavior, you can check for complex empty arrays using the `isreal` function and convert them to real empty arrays using the `real` function.

### Code Compatibility Report: Generate compatibility report from Current Folder browser

The Code Compatibility Report displays potential compatibility issues in your code and categorizes them by severity. This report helps you to update code to a newer MATLAB release by grouping issues into those issues that require immediate resolution and those issues you can address in a later release.

As of R2018a, you can generate the Code Compatibility Report from the Current Folder browser. Previously, you generated the report using the `codeCompatibilityReport` function only. For more information, see [MATLAB Code Compatibility Report](#).

### timer Object: Access properties with multilevel indexing

The `timer` object supports multilevel indexing for setting and accessing properties. For example, consider a timer `t` with a `UserData` property that contains a structure. Now you can access the data in the structure directly.

```
t.UserData.field1
```

To access the `field1` value in previous versions of MATLAB, you either defined an intermediate variable, or used the `getField` function.

```
d = t.UserData;  
d.field1
```

```
getField(t.UserData, 'field1')
```

Now you can set the values in the structure directly.

```
t = timer;  
t.UserData.field1 = 'value1';  
t.UserData.field2 = 'value2';
```

Using dot notation to access properties on an array of `timer` objects now returns a comma-separated list instead of a cell array. Using dot notation to set properties on an array of `timer` objects now errors.

## Version History

If your code relies on MATLAB returning a cell array when getting the properties for an array of `timer` objects, either update your code to work with a comma-separated list, or use the `get` function. This code continues to return a cell array.

```
t_arr = [timer timer timer];
get(t_arr,'Name')
```

```
ans =
```

```
3×1 cell array
```

```
{'timer-5'}
{'timer-6'}
{'timer-7'}
```

To continue to set properties for all elements in an array of `timer` objects, use the `set` function.

```
set(t_arr,'TimerFcn','myFunction')
```

In MATLAB R2018a and later, using dot notation to set the properties for all `timer` objects in an array results in an error. For example, `t_arr.TimerFcn = 'myFunction'` now errors.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
complex function and functions supporting complex input with the 'like' syntax	Still runs	Not applicable	Previously, the <code>complex</code> function and functions that support complex input with the 'like' syntax (such as <code>zeros</code> and <code>ones</code> ) always returned real empty arrays when one or more dimensions were 0. To preserve this behavior, you can check for complex empty arrays using the <code>isreal</code> function and convert them to real empty arrays using the <code>real</code> function.

Functionality	Result	Use This Instead	Compatibility Considerations
MATLAB path names containing '.', '..', and symbolic links	Still runs	Not applicable	<p>MATLAB now resolves all path names containing '.', '..', and symbolic links to their target location before adding them or removing them from the path. Resolving the path names ensures that each entry in the MATLAB path represents a unique folder location.</p> <p>For example, if you run the command <code>addpath('c:\matlab\..\work')</code>, MATLAB adds the folder <code>c:\work</code> to the path.</p> <p>MATLAB also resolves path names when changing the current folder.</p> <p>For more information, see <code>addpath</code>, <code>rmpath</code>, <code>path</code>, and <code>cd</code>.</p>
Case sensitivity when adding or removing folders to the MATLAB path.	Still runs	Not applicable	<p>MATLAB now case corrects path names according to the file system's case characteristics before adding or removing them from the path.</p> <p>For example, if the folder <code>c:\TEMP</code> exists on your system and you run the command <code>addpath('c:\temp')</code>, MATLAB adds the folder <code>'c:\TEMP'</code> to the path.</p> <p>For UNC paths, MATLAB case corrects only the file path part of the path. The server name and share name parts of the path are standardized to all lowercase.</p>

Functionality	Result	Use This Instead	Compatibility Considerations
ans in nested functions	Still runs	Not Applicable	<p>Currently, nested functions sometimes can access and modify the ans variable that is defined in the parent function. In future releases, nested functions will never have access to the ans variable defined in the parent function. For example in future releases, in this code, MATLAB throws an error in the nested function.</p> <pre data-bbox="914 575 1430 772">function foo     1+2 % Assigns to ans.     nested()     function nested         ans % MATLAB throws an error.     end end</pre> <p>To share a value between a nested function and its parent function, assign the value to a variable other than ans.</p> <pre data-bbox="914 919 1187 1117">function foo     sharedVar = 1+2     nested()     function nested         sharedVar     end end</pre>



Functionality	Result	Use This Instead	Compatibility Considerations
MATLAB classification of identifiers in anonymous functions	Still runs	Not applicable	<p>Currently, if any identifiers in an anonymous function are unknown at creation time, all identifiers are unknown. In a future release, they will be classified using the same rules as if they were inside the function. If an identifier is known in the function, it is known in the anonymous function. For example, consider the following function and its local function.</p> <pre data-bbox="914 604 1536 831"> function myfun myscript; % script sets x = 1 and lf = 10 f1 = @()lf(1); f2 = @()lf(x); end function lf(y) % local function to myfun end </pre> <p>All identifiers in <code>f1</code> are known when MATLAB creates the anonymous function. It calls the local function <code>lf</code> with an input of 1. In the current version of MATLAB, since <code>x</code> is unknown, MATLAB considers <code>lf</code> to be unknown also, and resolves both identifiers at runtime. Therefore, it uses <code>x</code> from <code>myscript</code> to index into the variable <code>lf</code> from <code>myscript</code>.</p> <p>In a future version of MATLAB, <code>lf</code> in <code>f2</code> will resolve to the local function, since <code>lf</code> is known when the anonymous function is created. <code>x</code> continues to be unknown until runtime. At runtime, MATLAB will use <code>x</code> from <code>myscript</code> as input to the local function <code>lf</code>, instead of as an index to the variable <code>lf</code> from <code>myscript</code>.</p>

Functionality	Result	Use This Instead	Compatibility Considerations
MATLAB resolution of identifiers within <code>parfor/spmd</code> blocks	Still runs	Not applicable	<p>Currently, MATLAB resolves identifiers in <code>parfor/spmd</code> blocks only within the scope of that block. In a future version of MATLAB, these identifiers will have the same resolution outside the <code>parfor/spmd</code> block. For example, consider the following function.</p> <pre>function myFunc()     parfor i = 1:2         y(i);     end     i end</pre> <p>Currently, inside the <code>parfor</code> block, <code>i</code> is recognized as a variable, but outside of the <code>parfor</code> block, it is an unrecognized function or variable. In a future version of MATLAB, <code>i</code> will be recognized as a variable outside of the <code>parfor</code> loop. However, its value from the <code>parfor</code> loop is not available outside of the block. In this example, the value of <code>i</code> is <code>sqrt(-1)</code> outside of the <code>parfor</code> loop.</p>
Change in precedence of compound name resolution	Still runs	Not applicable	<p>In MATLAB, a compound name is a name comprised of several parts joined by a dot. For example <code>structName.fieldName</code> or <code>packageName.ClassName.methodName</code>. In the current version of MATLAB, if a compound name, such as <code>a.b.c</code>, does not resolve to a variable, then it has the following precedence order.</p> <ol style="list-style-type: none"> <li>1 Class called <code>a.b.c</code></li> <li>2 Class called <code>a.b</code> with static method <code>c</code></li> <li>3 Package function called <code>a.b.c</code></li> <li>4 Class called <code>a.b</code> (no static method <code>c</code>)</li> <li>5 Class called <code>a</code></li> </ol> <p>In a future version of MATLAB, if <code>a.b.c</code> does not resolve to a variable, it will have this precedence order.</p> <ol style="list-style-type: none"> <li>1 Class or package function called <code>a.b.c</code></li> <li>2 Class or package function called <code>a.b</code></li> <li>3 Class or function called <code>a</code></li> </ol>
Deleting a folder that was removed outside of MATLAB	Warns	Not applicable	MATLAB now detects and warns if you try to delete a folder that was removed outside of MATLAB.

Functionality	Result	Use This Instead	Compatibility Considerations
notebook	Errors	Live Editor	notebook has been removed. To create a document that combines code, formatted text, and output, use the Live Editor instead.
Change in precedence of *based imports	Warns	To use a package function, use the fully qualified name.	<p>Imports in a function have the highest precedence. Imports shadow variables, local functions, and nested functions. For example,</p> <pre>function myfunc %import "local" and "nest" functions import pkg1.* local() % Calls "pkg1.local"         % and displays warning.  function nest end  nest(); % Calls "pkg1.nest"         % and displays warning. end  function local end</pre> <p>In the future, imported functions will have lower precedence than variables, nested functions, and local functions</p> <pre>function myfunc %import "local" and "nest" functions import pkg1.* local() % Calls "myfunc/local"  function nest end  nest(); % Calls "myfunc/nest" end  function local end</pre>

Functionality	Result	Use This Instead	Compatibility Considerations
Fully qualified import behavior	Warns	To use a function "nest" from a fully qualified import instead of the nested function with the same name, either remove the import statement or the nested function definition.	<p>A fully qualified import takes precedence over a nested function.</p> <pre>function myfunc % import function "nest" import pkg.nest % Calls "pkg.nest" and displays warning nest();  function nest end end myfunc</pre> <p>Warning: Prec-1.File: myfunc.m Line: 2 Column: 10 Using "nest" from the fully qualified import instead of the nested function with the same name. This will error in a future release. Either remove the import statement or the nested function definition.</p> <p>In the future, fully qualified imports sharing names with identifiers in the same scope will throw an error.</p>

Functionality	Result	Use This Instead	Compatibility Considerations
Fully qualified imports with names that shadow identifiers in outer scopes	Warns	Not applicable	<p>A fully qualified import is ignored when it shadows an identifier in the outer scope.</p> <pre> 1. function myfunc 2.   x = 1; 3.   function nest 4.     import pkg1.x % Calls variable "myfunc/x" on line 2 % and displays warning 5.     x() 6.   end 7. end  myfunc  Warning ID: mir_nr_id_shadows_imported_function Message: "x" is both the name of an imported function in a nested function and the name of a variable in the outer scope. The imported "x" function is not called. In future, the imported "x" will be called.  In the future, fully qualified imports will shadow identifiers in the outer scopes with same name.  1. function myfunc 2.   x = 1; 3.   function nest 4.     import pkg1.x 5.     x() % Calls imported function "pkg1.x" 6.   end 7. end </pre>
Error handling for fully qualified imports that cannot be resolved	Still runs	Not applicable	<p>If MATLAB is launched without Java, then fully qualified imports that cannot be resolved do not throw an error.</p> <p>If MATLAB is launched with Java, then fully qualified imports that cannot be resolved throw an error.</p> <p>In the future, fully qualified imports that cannot be resolved throw an error with or without Java.</p>

Functionality	Result	Use This Instead	Compatibility Considerations
Imports used in scripts compared with imports used in functions	Still runs	Not applicable	<p>If a script contains an import statement and that import statement matches the name of a class on the path, then all uses of that identifier resolve to the class. If the identifier does not resolve to a class at runtime, then an error is thrown.</p> <p>In the future, imports used in scripts will behave like imports in functions.</p>
Error handling when import not found	Still runs	Remove functions such as <code>javachk</code> and <code>usejava</code> used for error handling.	<p>Fully qualified imports that cannot be resolved throw an error in MATLAB -nojvm mode. For example, create this function.</p> <pre>function myfunc import java.lang.String if ~usejava('jvm') disp('This function requires Java'); else % do something with java String class end end</pre> <p>Start <code>matlab -nojvm</code>, and then run the function.</p> <pre>myfunc</pre> <p>This function requires Java</p> <p>In the future, MATLAB throws an error. For example:</p> <pre>myfunc</pre> <p>Unable to find class or function 'java.lang.String' (line 2) for import. If your class or function requires Java, restart MATLAB without the -nojvm option.</p>

## Mathematics

### graph and digraph Objects: Work with multigraphs that have multiple edges between two nodes

The `graph` and `digraph` classes now support multiple edges between two nodes. Repeated edges are distinct and can have different weights. To facilitate working with graphs that have repeated edges, several new functions are available:

- `ismultigraph` — Determine whether a graph is a multigraph or a simple graph.
- `simplify` — Reduce a multigraph into a simple graph. Optionally, you can specify a rule to combine or pick between repeated edges.
- `edgecount` — Count the number of edges between two nodes.
- `outedges`, `inedges` — Find outgoing or incoming edge indices for a particular node.

Additionally, some existing graph functions have updated capabilities to account for multiple edges.

Function	New Functionality
<code>shortestpath</code>	New third output that lists the edge indices of all edges on the path.
<code>shortestpathtree</code>	New third output that indicates whether each edge is in the tree.
<code>highlight</code>	All edges between the source and target nodes are highlighted by default. A new name-value pair 'Edges' enables you to highlight specific edges using indices, which is compatible with the new third output of <code>shortestpath</code> .
<code>bfsearch</code> , <code>dfsearch</code>	New second output that contains the edge index. If the primary output is a table, then it has an additional variable <code>EdgeIndex</code> with the same information.
<code>isomorphism</code>	New second output that contains permutation information of the edges.
<code>findedge</code>	New second output that contains edge indices.

### Version History

There are a few changes that might require updates to existing code:

- 1 neighbors function counts self-loops only once.** In previous releases, if node `u` had a self-loop, then `neighbors(g,u)` listed `u` twice in the output. `neighbors(g,u)` now returns only one instance of `u`.
- 2 Display change of self-loops in plot of a simple graph.** Self-loops in the plot of a simple graph are now shaped like a leaf or teardrop. In previous releases, self-loops were displayed as circles.
- 3 graph, digraph, and addedge no longer produce errors when they encounter duplicate edges.** Instead, the duplicate edges are added to the graph and the result is a multigraph. The

`ismultigraph` function is useful to detect this situation, and `simplify` provides an easy way to remove the extra edges.

## **graph and digraph Objects: Calculate component sizes and weighted adjacency matrices**

The `conncomp` function offers a second output to return the number of nodes in each connected component. Additionally, the `adjacency` function accepts a second input to create weighted adjacency matrices.

## **GraphPlot Object: Visualize graphs with additional options for 'force', 'force3', and 'circle' layouts**

When you plot a graph `G` with a specific layout using `plot(G, 'Layout', method)`, you can use a number of layout-specific name-value pairs for each of the different methods. For example, `plot(G, 'Layout', 'layered', 'Direction', 'left')` changes the orientation of a layered layout.

New layout-specific name-value pairs for visualizing directed and undirected graphs with the `plot` function include:

- `'WeightEffect'` name-value pair — To incorporate edge weights into the `'force'` and `'force3'` layouts.
- `'UseGravity'` name-value pair — To turn the effects of gravity on (set to off by default) with the `'force'` and `'force3'` layouts. When gravity is turned on, plotting a graph with multiple components attracts all of the nodes to the origin and enables large components to take up more space.
- `'Center'` name-value pair — To specify a single node that should appear in the center of a circular graph plot with the `'circle'` layout.

## **polyshape Objects: Analyze polygons with turningdist, nearestvertex, and overlaps functions**

New functionality for analyzing 2-D polygons is available for `polyshape` objects:

- `turningdist` function returns a number close to 0 when two input `polyshape` objects have nearly matching boundary shapes, regardless of scale or orientation.
- `nearestvertex` function returns the closest vertex of an input `polyshape` object to a query point.
- `overlaps` function determines if two `polyshape` objects are overlapping.

## **polyshape Objects: Return vertex map and accept arrays with compatible sizes for intersect, subtract, union, and xor functions**

The `intersect`, `subtract`, `union`, and `xor` functions for `polyshape` objects now return vertex mapping information and accept input arrays of `polyshape` objects with compatible sizes.

Vertex mapping enables you to identify where the vertices of an output `polyshape` originated. For example, `[pout, pshapeID, vertexID] = intersect(pshape1, pshape2)` returns column



vectors `pshapeID` and `vertexID` containing the same number of rows as the number of vertices in `pout`.

Each element of `shapeID` contains the value 1, 2, or 0:

- Element is 1 when the corresponding vertices of `pout` belong to `pshape1`.
- Element is 2 when the corresponding vertices of `pout` belong to `pshape2`.
- Element is 0 when the corresponding vertices of `pout` belong to the intersection of `pshape1` and `pshape2`.

The vertex information `vertexID` contains the row numbers for the corresponding vertices in `pshape1` and `pshape2`.

### **polybuffer Function: Create buffer around points or lines**

The `polybuffer` function enables you to specify a buffer around a set of points or line segments. For example, `pshape = polybuffer([0 0; 2 3], 'Points', 1)` creates a `polyshape` object `pshape` whose boundaries are circles of radius 1 centered about the points (0,0) and (2,3).

### **triangulation Objects: Find neighboring vertices and locations of query points with improved performance**

You can compute neighboring vertices and locations of query points faster with the `nearestNeighbor` and `pointLocation` functions for `triangulation` objects.

### **ode45 Function: Solve nonstiff differential equations faster**

The `ode45` function shows improved performance for some problems.

## Graphics

### **Axes Object: View axes at small size with improved layout, limit selection, and font scaling**

Small axes have an improved layout that reduces white space for better readability. To reduce white space, the axes limits now fit more tightly around the data. Also, legends and colorbars have narrower margins. Additionally, the font size scales down slightly for small axes to avoid overlapping text or text that runs off the figure.

### **Version History**

For small axes, you might encounter slightly different limits, tick values, or font sizes than in previous releases. However, if you previously specified the limits, tick values, or font size, then the axes still uses the values that you specify. To specify the limits, use the `xlim` and `ylim` functions. To specify the tick values, use the `xticks` and `yticks` functions. To specify the font size, use the `FontSize` property.

### **Axes Object: Map data values to colormap using linear or logarithmic scale**

Scale colormaps linearly or logarithmically using the new `ColorScale` property for Axes objects.

### **Legend Object: Create legends with multiple columns**

You can create legends with multiple columns using the new `NumColumns` property for Legend objects. For an example, see [Add Legend to Graph](#).

### **heatmap Function: Zoom and pan data, display data tips, and sort rows and columns interactively**

Heatmaps have new options for interacting with data:

- Zoom — Use the scroll wheel or the `+` and `-` keys to zoom.
- Pan — Use the arrow keys to pan across the rows or columns.
- Data tips — Hover over the heatmap to display a data tip.
- Rearrange rows and columns — Click and drag a row or column label to move it to a different position.
- Sort values — Click the icon that appears when you hover over the row or column label. Click once to sort the values in ascending order, twice to sort the values in descending order, and a third time to reset the order.

### **geobubble Function: Explore with interactive data tips and a scale bar**

Geographic bubble charts now support data tips. Hover over a bubble on the chart to display a data tip.

In addition, geographic bubble charts now include a scale bar in their lower left corner to indicate the actual distances represented in the map. Use the `ScalebarVisible` property to control whether your chart includes a scale bar.

## Axes Toolbar: Add toolbars to your axes for quick access to pan, zoom, and other data exploration tools

You can add a toolbar to the top-right corner of the axes for quick access to data exploration tools. The toolbar typically includes options to brush data, add data tips, rotate (3-D axes only), pan, and zoom.



To add the toolbar, use the new `Toolbar` property for `Axes` objects. The `Toolbar` property stores an `AxesToolbar` object. Set the `Visible` property of the `AxesToolbar` object to `'on'`. Some of the axes toolbar options are redundant with the figure toolbar. You can remove the redundant options from the figure toolbar using the `removeToolbarExplorationButtons` command.

```
ax = gca;
ax.Toolbar.Visible = 'on';
removeToolbarExplorationButtons(gcf);
```

To restore the figure toolbar exploration buttons, use the `addToolbarExplorationButtons(gcf)` command.

## Property Inspector: Modify graphics interactively with an improved property inspector

An improved **Property Inspector** makes it easier to modify graphics objects interactively. To open the inspector, use the `inspect` function. For example, `inspect(gca)` opens the inspector for the current axes.

## Polygon Object: Control color and transparency of hole edges

You can control the color and transparency of the polygon hole boundaries. Use the new `HoleEdgeColor` and `HoleEdgeAlpha` properties for `Polygon` objects.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Changing the figure colormap affects all axes in the figure	Still runs	Not applicable	<p>When you change a figure's colormap, the colormap changes for all axes in the figure. Previously, if you specified the colormap for a particular axes, then changing the figure's colormap did not affect the axes.</p> <p>Set the axes colormap after setting the figure colormap to avoid the figure's colormap overwriting the axes colormap.</p>
Axes limits, tick values, or font sizes for small axes might be different	Still runs	Not applicable	<p>Due to an improved layout that reduces white space, small axes might have slightly different limits, tick values, or font sizes than in previous releases.</p> <p>If you prefer the limits, tick values, or font sizes from previous releases, you can manually set them. To specify the limits, use the <code>xlim</code> and <code>ylim</code> functions. To specify the tick values, use the <code>xticks</code> and <code>yticks</code> functions. To specify the font size, use the <code>FontSize</code> property.</p>
Setting title properties of geographic bubble chart to empty ([]) array	Still runs	Not applicable	<p>GeographicBubbleChart objects now store empty title properties as an 0-by-0 empty character array. The title properties include the <code>Title</code>, <code>SizeLegendTitle</code>, and <code>ColorLegendTitle</code> properties.</p> <p>Previously, setting the property to an empty array either returned an error or stored the value with unexpected dimensions (as a 0-by-0-by-0 empty character array).</p>
Setting title properties of geographic bubble chart to multiline title	Still runs	Not applicable	<p>GeographicBubbleChart objects now store multiline titles as a cell array of character vectors. The title properties include the <code>Title</code>, <code>SizeLegendTitle</code>, and <code>ColorLegendTitle</code> properties.</p> <p>Previously, the chart stored the value as a character matrix and displayed the title with incorrect alignment.</p>

## Data Import and Export

### **readtable Function: Specify the number of rows to read from a text file using import options**

You can select a subset of rows to read from a text file using `readtable` along with import options. Use the `DataLines` property to specify the rows to read. Specify `DataLines` in multiple ways:

- `opts.DataLines = 5` sets the `DataLines` property to the value `[5 inf]`. The `readtable` function reads all rows of data starting from row 5 to the end-of-file.
- `opts.DataLines = [2 6]` sets the property to read lines 2 through 6. The `readtable` function reads all rows of data starting from row 2 to row 6.
- `opts.DataLines = [1 3; 5 6; 8 inf]` sets the property to read rows 1, 2, 3, 5, 6, and all rows between 8 and the end-of-file.

After specifying `DataLines`, import the data using `readtable`:

```
T = readtable(filename,opts);
```

For more information, see the `DataLines` property of `DelimitedTextImportOptions` and `FixedWidthImportOptions` objects.

### **readtable Function: Easily manage prefixes and suffixes from data using import options**

You can specify characters to remove from the prefix and suffix positions in a variable using `readtable` with import options. Specify the prefix and suffix options in multiple ways:

- `opts = setvaropts(opts, 'Var1', 'Prefixes', '$')` sets the `Prefixes` option for the variable `Var1`. If `Var1` contains a value of '\$500', then `readtable` reads it as '500'.
- `opts = setvaropts(opts, 'Var1', 'Suffixes', '/-')` sets the `Suffixes` option for the variable `Var1`. If `Var1` contains a value of '\$500/-', then `readtable` reads it as '\$500'.
- `opts = setvaropts(opts, 'Var1', 'TrimNonNumeric', true)` sets the `TrimNonNumeric` option for variable `Var1`. If `Var1` contains a value of '\$500/-', then `readtable` reads it as 500.

For more information, see the variable options and descriptions on the `setvaropts` reference page.

### **preview Function: Preview first 8 rows of a table in a file without importing the full table**

You can preview the first 8 rows of a table from a file before importing the full table using `readtable` and import options. For more information, see the `preview` function.

### **imageDatastore Function: Work with millions of images with improved memory usage and performance**

The `imageDatastore` function now supports faster processing of millions of images for deep learning applications with improved memory usage.

## **Datastore Functions: Seamlessly work with datasets stored on cloud and local machines**

You can work with datasets stored on cloud and local machines by using the 'AlternateFileSystemRoots' parameter in datastore functions and the File-set object for custom datastores. This parameter is supported for:

- Datastore objects: TabularTextDatastore, SpreadsheetDatastore, ImageDatastore, FileDatastore, KeyValueDatastore, and TallDatastore.
- File-set object: `matlab.io.datastore.DsFileSet` for custom datastores.

For more information, see [Set Up Datastore for Processing on Different Machines or Clusters](#).

## **Datastore Functions: Read HDFS data more easily when using Hortonworks or Cloudera**

You can use datastore functions to access HDFS data more easily on Hortonworks or Cloudera® without having to set the HADOOP\_HOME or HADOOP\_PREFIX environment variables. MATLAB now automatically assigns these environment variables when using Hortonworks or Cloudera application edge nodes.

For more information, see [Read Remote Data](#).

## **readtable, detectImportOptions, datastore, and tabularTextDatastore Functions: Automatically detect and return duration data in text files**

The functions `readtable`, `detectImportOptions`, `datastore`, and `tabularTextDatastore` detect and return duration data as duration type. For more information on duration arrays, see the duration reference page.

### **Version History**

Previously, `readtable`, `detectImportOptions`, `datastore`, and `tabularTextDatastore` functions returned duration data as text. To preserve that behavior, use the `DurationType` parameter. For example, `T = readtable('myfile.txt', 'DurationType', 'text')` reads duration data in `myfile.txt` as text. For more information on the `DurationType` parameter, see the function reference pages.

## **detectImportOptions Function: Control import properties of duration data**

The `detectImportOptions` now can detect and manage variables of type duration in tabular data.

For a variable in a table containing duration data, you can use the `setvaropts` function to set these properties: `DurationFormat`, `InputFormat`, `DecimalSeparator`, and `FieldSeparator`. For more information, see the `setvaropts` function page.

## VideoReader Function: Read video files faster on all platforms

VideoReader has improved performance for all supported video formats when you read video frames in a loop as part of a general video processing workflow. For more information on supported video formats and function usage, see [VideoReader](#).

## VideoWriter Function: Write video files faster on all platforms

VideoWriter has improved performance for all supported video formats, when you write video frames in a loop as part of a general video processing workflow. For more information on supported video formats and function usage, see [VideoWriter](#).

## openDiskFile Function: Read data files in FITS (Flexible Image Transport System) data format

Read data files in FITS (Flexible Image Transport System) data format using the `openDiskFile` function.

## webwrite Function: Support for NTLM authentication

The `webwrite` function and `weboptions` POST and PUT methods support NTLM authentication on Windows platforms.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
DataLine property of DelimitedTextImportOptions and FixedWidthImportOptions objects	Still runs	DataLines	Use property name DataLines instead of DataLine.
readall method for SpreadsheetDatastore, TabularTextDatastore, KeyValueDatastore, and TallDatastore objects	Still runs	Not applicable	<p>The <code>readall(ds)</code> method no longer resets the datastore <code>ds</code> to a state where no data has been read from it. This new behavior of <code>readall</code> enables the <code>read</code> method to read data from the end of the previous <code>read</code> operation.</p> <p>Previously, <code>readall(ds)</code> would reset the datastore specified by <code>ds</code> to the state where no data has been read from it. Therefore, any call to <code>read</code> after a <code>readall</code> would always read from the beginning of the datastore.</p>

Functionality	Result	Use This Instead	Compatibility Considerations
Output from sheetnames function	Still runs	Not applicable	<p>The sheetnames function now returns the names argument as a string array.</p> <p>Previously, sheetnames returned names as a cell array of character vectors. To convert names back to a cell array of character vectors, use the following:</p> <pre>% Get sheet names from a file 'myfile3.xls' % in the spreadsheetDatastore ssds names = sheetnames(ssds,'myfile3.xls'); % Convert names to cell array of character vectors names = cellstr(names);</pre>
hdfgd	Errors	matlab.io.hdfeos.gd	Replace all instances of hdfgd with the corresponding function in the matlab.io.hdfeos.gd package.
hdfsd	Errors	matlab.io.hdf4.sd	Replace all instances of hdfsd with the corresponding function in the matlab.io.hdf4.sd package.
hdfsw	Errors	matlab.io.hdfeos.sw	Replace all instances of hdfsw with the corresponding function in the matlab.io.hdfeos.sw package.



## Data Analysis

### **groupsummary Function: Group and discretize data for summary operations on table and timetable variables**

Grouping and discretizing table and timetable data for summary operations is now easier with the `groupsummary` function. For example, `groupsummary(T, 'Gender', 'method', 'mean')` computes the mean of the variables in a table `T` by gender.

### **Table and Timetable Variables: Add, delete, and rearrange column-oriented variables with the functions `addvars`, `removevars`, `movevars`, `splitvars`, `mergevars`, `rows2vars`, and `inner2outer`**

Function	Description
<code>addvars</code>	Add variables to table or timetable.
<code>removevars</code>	Delete variables from table or timetable.
<code>movevars</code>	Move variables in table or timetable.
<code>splitvars</code>	Split multicolumn variables in table or timetable.
<code>mergevars</code>	Combine table or timetable variables into multicolumn variable.
<code>rows2vars</code>	Reorient rows of table or timetable so that rows become variables.
<code>inner2outer</code>	Invert a nested table-in-table hierarchy.

### **Preallocated Tables and Timetables: Initialize table and timetable variables so that they have specified sizes and data types**

To initialize table and timetable variables so that they have specified sizes and data types, specify the `'Size'` and `'VariableTypes'` name-value pair arguments of the `table` and `timetable` functions. You can specify the number of rows, the number of variables, and the data types of the variables. The preallocated variables contain default values or empty arrays, depending on the specified data types. You can fill the variables with data values later.

### **Regular Timetables: Create regularly spaced timetables using a time step or sampling rate**

You can create regular timetables using either a time step or a sampling rate to specify identical time intervals between consecutive row times. To create regular timetables, use the `timetable` or `array2timetable` functions with the `'TimeStep'` or `'SamplingRate'` name-value pair arguments. You also can specify the value of the first row time using the `'StartTime'` name-value pair.

## retime and synchronize Functions: Synchronize timetables to a time step or sampling rate that you specify

To synchronize timetables to a time step or sampling rate, use the 'TimeStep' or 'SamplingRate' name-value pair arguments of the `retime` or `synchronize` functions.

## duration Arrays: Create duration arrays from text that represents elapsed times

You can convert text representing elapsed times into a duration array using the `duration` function. The input text represents each time in a format such as 'hh:mm:ss' or 'dd:hh:mm:ss'. The fields dd, hh, mm, and ss represent days, hours, minutes, and seconds, respectively.

## normalize Function: Normalize array, table, and timetable data

You can normalize data in an array, table, or timetable by quantities such as the z-score or p-norm using the `normalize` function. For example, `normalize(A, 'norm', 1)` normalizes each column of a matrix `A` by its 1-norm.

## tall Arrays: Operate on tall arrays with more functions, including smoothdata, find, and isoutlier

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, type `help tall/functionName`. For example, `help tall/find`.

<code>array2timetable</code>	<code>isoutlier</code>
<code>caldiff</code>	<code>isprotected</code>
<code>filloutliers</code>	<code>maxk</code>
<code>find</code>	<code>mink</code>
<code>flip</code>	<code>rescale</code>
<code>fliplr</code>	<code>smoothdata</code>
<code>iscategory</code>	<code>splitlines</code>
<code>isordinal</code>	

## tall Array Indexing: Use tall numeric arrays to index the first dimension

For a tall array `X`, you can index the first dimension of the array with `X(idx,...)`, where `idx` is a tall numeric array.

## tall Arrays: Solve linear systems $Ax = b$

You can solve linear systems  $Ax = b$  with a tall coefficient matrix `A` by `x = A\b`. The solver for tall arrays uses a QR decomposition to find a least-squares solution to the problem.

## tall Arrays: Return group labels with findgroups

findgroups now supports multiple output arguments with tall arrays. The additional outputs from findgroups contain unique lists of group labels for each grouping variable.

## tall Arrays: Set date and time components of tall datetime and tall duration arrays

You can change the properties of tall datetime and tall duration arrays using dot indexing. For example, `t.Format = 'dd-MMM-yyyy'` changes the display format of `t`.

See the `datetime` and `duration` pages for a list of the properties, or `Extract or Assign Date and Time Components of Datetime Array` for examples.

## tall Arrays: Set properties of tall tables and tall timetables

You can change the properties of tall tables and tall timetables using dot indexing. For example, `T.Properties.VariableNames = {'Name1' 'Name2'}` changes the variable names of `T`. For a list of all properties use the command `T.Properties`.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Adding and subtracting durations represented as text in timer format from a datetime array	Still runs	Not applicable	<p>Starting in R2018a, if you represent a duration as text, you can add and subtract it from a datetime array, and the result is a datetime array. In both operations, text in timer format is converted to a duration value. For example, this code adds 1 hour and 30 minutes to the current date and time.</p> <pre>datetime('now') + '1:30:00'</pre> <p>This code subtracts 12 hours from the current date and time.</p> <pre>datetime('now') - '12:00:00'</pre> <p>Previously, if you represented a duration as text in a timer format, such as <code>'1:30:00'</code>, then adding that duration to a datetime array raised an error, while subtracting it returned a duration array, instead of a datetime array.</p> <p>To reproduce the earlier behavior, convert the text to a datetime value. Subtracting a datetime array from another datetime array returns a duration array.</p> <pre>datetime('now') - datetime('12:00:00')</pre>

Functionality	Result	Use This Instead	Compatibility Considerations
Naming the row times vector of a timetable when you specify the 'VariableNames' name-value pair argument of the <code>timetable</code> function	Still runs	Not applicable	<p>Starting in R2017b, the <code>timetable</code> function assigns the name of the input time vector as the name of the row times vector, even when you specify variable names using the 'VariableNames' name-value pair argument. Previously in R2016b and R2017a, <code>timetable</code> assigned 'Time' as the name of the row times vector when you specified 'VariableNames'. This behavior changed because the row times vector is not a timetable variable. The row times are the labels of the rows.</p> <p>To rename the row times vector, use the <code>DimensionNames</code> property. For example, <code>TT.Properties.DimensionNames{1} = 'Time'</code> changes the name of the row times vector of <code>TT</code> to 'Time'.</p>
Subscripting into a timetable using datetime or duration values as row subscripts	Still runs	Not applicable	<p>Starting in R2018a, timetable subscripting uses a tolerance to match subscripts to row times. Therefore, one subscript might match two or more distinct row times in a timetable when those row times differ only by a small amount. Previously in R2016b through R2017b, you could access data in a timetable using a datetime or duration array as the row subscript, but subscripting required an exact match between a time in the subscripts and a row time of the timetable.</p> <p>You can reproduce the earlier behavior that required an exact match in several ways. To select only exact matches at a single time, create a logical row subscript by testing for equality between subscripts and the row times using the equality operator, <code>==</code>. To select only exact matches at multiple times, create a logical row subscript using the <code>ismember</code> function. As an alternative, use the <code>withtol</code> function with a tolerance of seconds (0).</p>

## App Building

### **App Designer: Create deployed web apps using MATLAB Compiler**

If you have MATLAB Compiler installed on your system, you can create web apps using the **Share** button in the App Designer toolstrip. For more information, see [Ways to Share Apps](#).

Google Chrome and Internet Explorer (version 11 and later) support running web apps.

### **App Designer: Add and configure tree components on the App Designer canvas**

Drag and drop tree components from the **Component Library** onto the canvas, and change their text labels and positions directly in the canvas.

### **App Designer: Select from recently used argument sets when running apps with input arguments**

Configure up to seven sets of input arguments under the **Run** button in the App Designer toolstrip. Select any of the sets to run them again. The sets remain in the drop-down list until you close the app.

### **App Designer: Edit axes title and label directly in the canvas**

Now you can modify plot titles and axes labels more quickly by editing them directly on the canvas.

### **GUIDE: Migrate GUIDE apps to App Designer**

For assistance in migrating your apps to App Designer, use the [GUIDE to App Designer Migration Tool for MATLAB](#). This tool is available as a support package.

### **App Testing Framework: Author automated tests for App Designer apps**

Use the app testing framework to write automated tests for your apps. The app testing framework leverages the MATLAB unit testing framework.

You can use the MATLAB app testing framework to test apps built with App Designer or apps built programmatically using the `ui figure` function. The app testing framework enables you to author a test class that programmatically performs a gesture on a UI component, such as pressing a button or dragging a slider, and verifies the behavior of the app.

For more information, see [App Testing Framework](#).

### **Figure Objects: Maximize and minimize figures programmatically**

Use the `WindowState` property to maximize, minimize, or display a figure in full-screen mode.

## **uitable Function: Specify data as table array**

The `Data` property of Table UI components now accepts table arrays. For example:

```
T = readtable('patients.xls');  
uit = uitable(uifigure,'Data',T);
```

table arrays are supported only in App Designer apps or figures created with the `uifigure` function.

## **uidatepicker Function: Add date selection controls to apps**

Call the `uidatepicker` function to add a date picker to an app.

To display a date picker in an App Designer app, call the `uidatepicker` function from within a callback, such as the `StartupFcn` for the `UIFigure` component.

Date pickers work only in App Designer apps or in figures created with the `uifigure` function.

## **uiprogressdlg Function: Create modal in-app progress dialog boxes to apps**

Call the `uiprogressdlg` function to create a progress dialog box within an app.

The dialog box can only display in App Designer apps or in figures created with the `uifigure` function.

## **uitree Function: Create trees with editable node text in the running app**

Specify the `Editable` property of a tree so that users can change the node text while the app is running. Specify the `NodeTextChangedFcn` callback to make the app respond when the user changes node text.

## **Component Text Alignment: Improved text alignment for labels, check boxes, and radio buttons**

The default height and vertical alignment for labels, check boxes, and radio buttons are now consistent with other single-line, text-based components. The new values make it easier to align the text of these components with other components.

Only labels, check boxes, and radio buttons in App Designer and in figures created with the `uifigure` function are affected.

## **Version History**

The new default height for these components is 22 pixels (previously, it was 15 pixels). Text is now centered vertically within the component's text box (previously, text was aligned to the top of the text box).

The labels, check boxes, and radio buttons in apps created in previous releases might look different when you open or run them in R2018a. You might need to change the vertical alignment, or adjust the height and location of those components to maintain a consistent appearance.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Apps saved in App Designer using R2018a	Error (when opening in previous releases)	Select <b>Save &gt; Save Copy As</b> in the R2018a App Designer toolstrip.	The .mlapp file format changed in R2018a. Apps created or modified in App Designer return an error when you try to open them in previous releases. To modify an app across multiple releases, select <b>Save &gt; Save Copy As</b> in the R2018a App Designer toolstrip. Then select the appropriate MATLAB release from the <b>Save as type</b> drop-down list. Alternatively, make a backup copy of the app before opening the app in R2018a.
Apps created in previous releases that contain labels, check boxes, or radio buttons	Still Runs	Not applicable	The new default height and vertical alignment of text for labels, check boxes, and radio buttons has changed. You might need to change the vertical alignment, or adjust the height and location of those components to maintain a consistent appearance. For more information, see “Component Text Alignment: Improved text alignment for labels, check boxes, and radio buttons” on page 9-30.

## Performance

### **Startup: Increased speed of MATLAB startup time**

MATLAB starts faster because of infrastructure improvements and optimizations.

### **Execution Engine: Execute tight loops with scalar math faster**

Loops that contain mainly indexing and scalar math operations execute faster due to execution engine optimizations.

### **Execution Engine: Improved performance for common programming patterns**

Common programming patterns, such as the evaluation of anonymous functions, and common combinations of operations, such as `isequal(size(x),size(y))`, execute faster due to targeted optimizations.

### **App Designer: Starting, loading, and layout tasks are faster**

Starting App Designer is 40% faster than in R2017b. Loading apps and performing layout tasks in App Designer are 10% to 30% faster than in R2017b.



## Hardware Support

### **Raspberry Pi: Support for Raspberry Pi Zero W board**

You can now use the MATLAB Support Package for Raspberry Pi Hardware with the Raspberry Pi Zero W board.

MATLAB Support Package for Raspberry Pi Hardware enables you to communicate with Raspberry Pi Zero W using Wi-Fi® or a micro-USB cable connection. For more information on how to communicate with Raspberry Pi Zero W, follow the steps in Install, Update, or Uninstall Support Package (MATLAB Support Package for Raspberry Pi Hardware).

### **MATLAB Online: Acquire live images from USB webcams in MATLAB Online**

You now can connect to and stream images from your USB webcam in MATLAB Online. Connect a USB webcam to your computer and give your browser access to your camera. All the functions in the MATLAB Support Package for USB Webcams are available for MATLAB Online.

## Advanced Software Development

### **Tab Completion: Describe your function syntaxes for custom tab completion and other contextual suggestions**

To customize code suggestions and completions for your functions, provide MATLAB with a JSON-formatted file that contains information about your function signatures. MATLAB uses this information to improve interactive features, such as tab completion and function hints. For more information, see [Customize Code Suggestions and Completions](#).

### **Unit Testing Framework: Run tests from the MATLAB Editor toolstrip**

You can run tests from the MATLAB Editor toolstrip. When you open a function-based or class-based test file, the Editor toolstrip has options to run all tests in the file or to run a single test in the file. Also, you can customize the test run with options, such as running tests in parallel (which requires Parallel Computing Toolbox) or running tests with a specified level of output detail.

The Run Tests controls in the toolstrip provide an alternative to programmatically running tests with the `runtests` function. For more information, see [Run Tests in Editor](#).

### **App Testing Framework: Author automated tests for App Designer apps**

Use the app testing framework to write automated tests for your apps. The app testing framework leverages the MATLAB unit testing framework.

You can use the MATLAB app testing framework to test apps built with App Designer or apps built programmatically using the `uifigure` function. The app testing framework enables you to author a test class that programmatically performs a gesture on a UI component, such as pressing a button or dragging a slider, and verifies the behavior of the app.

For more information, see [App Testing Framework](#).

### **Unit Testing Framework: Rerun failed tests with one click**

If a test failure is caused by incorrect or incomplete code for a test or for the code under test, it is useful to be able to rerun failed tests quickly and conveniently. If test failures exist in your test results, then MATLAB displays a link to rerun failed tests after it displays the number of failed tests.

```
Totals:  
  1 Passed, 1 Failed (rerun), 0 Incomplete.  
  0.25382 seconds testing time.
```

For more information, see [Rerun Failed Tests](#).

### **Unit Testing Framework: Test if values point to existing files or folders with `IsFile` and `IsFolder` constraints**

You can test if a value, specified as a string scalar or character vector, points to an existing file or folder.

- To test if a value points to an existing file, use `verifyThat`, `assertThat`, `assumeThat`, or `fatalAssertThat` with the `IsFile` constraint. For more information, see `matlab.unittest.constraints.IsFile`.
- To test if a value points to an existing folder, use `verifyThat`, `assertThat`, `assumeThat`, or `fatalAssertThat` with the `IsFolder` constraint. For more information, see `matlab.unittest.constraints.IsFolder`.

## Unit Testing Framework: Test if two sets are the same with `IsSameSetAs` constraint

To test if two sets are the same, use the `IsSameSetAs` constraint. The testing framework considers tests to be the same if they contain the same elements. Sets can be the same even if they have different order, shape, and size. For example, if `S = {'a' 'b' 'c'}`, then set `S` is the same as the following sets.

```
S1 = {'a' 'b' 'c'};
S2 = {'c'; 'a'; 'b'};
S3 = {'a' 'b' 'c' 'c' 'b'};
```

For more information, see `matlab.unittest.constraints.IsSameSetAs`.

## Unit Testing Framework: Select tests by test class hierarchy

You can run tests that have a specified superclass:

- To select and run test elements that have a specified superclass, use the `'Superclass'` name-value pair with the `runtests`, `runperf`, and `testsuite` functions or the `matlab.unittest.TestSuite` suite creation methods.
- To select test elements from an existing test suite, use the `TestSuite.selectIf` method with the `HasSuperclass` selector.

## Version History

Test suite elements created prior to MATLAB R2018a cannot be filtered by test class hierarchy. To filter these test elements by superclass, recreate the test suite in R2018a or later.

## Unit Testing Framework: Direct output stream to unique files for plugins

To direct text output from plugins to a unique file, use the `matlab.unittest.plugins.ToUniqueFile` output stream. This output stream is useful for running tests in parallel while redirecting output to a file, since it uses a unique file name for each instance of the output stream. The `ToUniqueFile` output stream is different from the `ToFile` output stream, which overwrites the file. For more information, see `matlab.unittest.plugins.ToUniqueFile`.

The `ToUniqueFile` output stream is supported for

- `matlab.unittest.plugins.DiagnosticsValidationPlugin`
- `matlab.unittest.plugins.FailureDiagnosticsPlugin`

- `matlab.unittest.plugins.TAPPlugin`
- `matlab.unittest.plugins.TestRunProgressPlugin`

Additionally, the `matlab.unittest.plugins.OutputStream` class provides an interface for test authors to create custom output streams.

## Unit Testing Framework: Increased access to parameterized testing properties

When you create a parameterized test, you define properties in a `properties` block with a `TestParameter`, `MethodSetupParameter`, or `ClassSetupParameter` attribute, depending on the parameterization level for your test. You can now access “up-level” parameters.

- Tests in a `Test` method block can access parameters that you define in `TestParameter`, `MethodSetupParameter`, and `ClassSetupParameter` properties blocks.
- Tests in a `TestMethodSetup` method block can access parameters that you define in `MethodSetupParameter` and `ClassSetupParameter` properties blocks.
- Tests in a `TestClassSetup` method block can access parameters that you define in a `ClassSetupParameter` properties block.

In previous versions of MATLAB, you could access only parameters from tests in method blocks at the corresponding level. For example, parameters defined in a `MethodSetupParameter` property block were only accessible from tests in a `TestMethodSetup` method block. For more information, see [Create Advanced Parameterized Test](#).

## Unit Testing Framework: Compare cell arrays of character arrays using `StringComparator`

In addition to supporting strings and character arrays, the `StringComparator` constraint now supports cell arrays of character arrays. For more information, see `matlab.unittest.constraints.StringComparator`.

## Version History

Update any tests that rely on the `StringComparator` constraint not being satisfied by equal cell arrays of character arrays. If the cell arrays of character arrays are equal, the comparator is satisfied. In previous versions of MATLAB, the comparator was not satisfied for any values that were cell arrays of character arrays. For example, the following test passes in MATLAB R2018a and later. In earlier versions of MATLAB, the test fails because the comparator did not support cell arrays of character arrays.

```
import matlab.unittest.TestCase
import matlab.unittest.constraints.StringComparator
import matlab.unittest.constraints.IsEqualTo

testCase = TestCase.forInteractiveUse;
actVal = {'coffee', 'cream', 'sugar'};
expVal = {'coffee', 'cream', 'sugar'};

testCase.verifyThat(actVal, IsEqualTo(expVal, 'Using', StringComparator))
```

## Unit Testing Framework: Comparison method for objects changed

As of R2018a, the `ObjectComparator` is satisfied if `isequaln` returns true. However, if the class of the expected value defines an `isequal` method, whether visible or hidden, but not an `isequaln` method, the `ObjectComparator` uses the `isequal` method for comparison. In previous releases, `ObjectComparator` used `isequal` to compare all objects unless the class of the expected value defined a visible `isequaln` method.

The `IsEqualTo` constraint and the `assertEqual`, `assumeEqual`, `fatalAssertEqual`, and `verifyEqual` qualification methods leverage `ObjectComparator` and, therefore, inherit the same change in behavior. For more information, see `matlab.unittest.constraints.ObjectComparator`.

## Version History

The `ObjectComparator` now uses `isequaln` for comparison instead of `isequal` if the class of the expected value

- Defines `isequaln`, whether visible or hidden.
- Does not define either `isequal` or `isequaln`.

## Performance Testing Framework: Define multiple, labeled measurement boundaries in test methods

Measurement boundaries enable you to refine which code the performance framework measures, and now you can label the measurement boundaries. Measurements from multiple boundaries with the same label in the same test method are accumulated and summed. The performance framework appends the label to the test element name in the measurement results. For more information, see `startMeasuring`.

## Mocking Framework: Specify default property values on mock object

When you create a mock object for a class, you can specify default values for properties that are implemented by the class. To specify default property values, use the `DefaultPropertyValues` name-value pair argument with the `createMock` method. For more information, see `matlab.mock.TestCase.createMock`.

## Mocking Framework: Obtain interaction history for mock object

To obtain a history of recorded interactions for a mock object, use either the `matlab.mock.TestCase.getMockHistory` method or the `matlab.mock.InteractionHistory.forMock` method. When you call certain publicly visible methods or access or modify certain publicly visible properties on a mocked class, the mocking framework records the interaction. For more information, see `matlab.mock.TestCase.getMockHistory` or `matlab.mock.InteractionHistory.forMock`.

## Mocking Framework: Construct mocks for classes that have Abstract properties with other attributes

You can construct a mock object for classes that have `Abstract` properties and other attributes. For example, you can construct a mock for a property that has `Abstract` and `Constant` attributes. The

mock implements the property as a concrete, Constant property. Similarly, properties with Abstract and Hidden attributes are implemented as concrete and Hidden properties. For more information, see Create Mock Object.

## **matlab.net.http Package: Stream data to and from a web service and handle forms and multipart messages**

To act on or display streamed data while it is being received, use the `matlab.net.http.io.ContentConsumer` class. To obtain or generate the data at the same time it is being sent, use the `matlab.net.http.io.ContentProvider` class. Using this class avoids the need to have all the data in memory before the start of a message,.

To handle forms and multipart messages, use the `matlab.net.http.io.MultipartConsumer` and `matlab.net.http.io.MultipartProvider` classes.

The following classes and methods were added to the `matlab.net.http` and `matlab.net.http.field` packages.

- `matlab.net.http.HeaderField` methods — `addFields`, `changeFields`, `getFields`, `removeFields`, `replaceFields`
- `matlab.net.http.field.GenericField` methods — `getParameter`, `removeParameter`, `setParameter`
- `matlab.net.http.field.ContentDispositionField` — This class specifies a Content-Disposition header field, which is commonly used in multipart form requests.
- `matlab.net.http.field.GenericParameterizedField` — This class is a version of the `GenericField` class that supports parameterized syntax.

## **C++ MEX Interface: Access MATLAB data and objects easier from C++**

Author MEX functions using modern C++ design patterns, extended data type support, and MATLAB copy-on-write semantics for faster handling of large data arrays. For more information, see C++ MEX Applications.

If you do not need MEX files that work in R2017b and earlier and you are familiar with modern C++, then consider using the new C++ MEX API and MATLAB Data API. If you are more comfortable working in the C language, continue using the C MEX API and C Matrix API.

## **Class Constructors: Author subclass without implementing a constructor solely to pass arguments through to a superclass constructor**

MATLAB passes arguments implicitly from a default subclass constructor to the superclass constructor. This behavior eliminates the need to implement a constructor method for a subclass only to pass arguments to the superclass constructor. For more information, see Implicit Call to Inherited Constructor.

## **Property Validation: Get information about property validation**

Get information on the validation defined for a property by accessing the validation metadata for that property. For more information, see Metadata Interface to Property Validation.

## Property Validation: Define validation for abstract properties

As of MATLAB Version 9.4 (R2018a), you can define validation for abstract properties. The validation applies to all subclasses. For more information, see Abstract Property Validation.

## Functions: Call numArgumentsFromSubscript for object dot method from overloaded subsref

In releases prior to MATLAB Version 9.4 (R2018a), the built-in numArgumentsFromSubscript function handled method calls of the form *object.method* incorrectly.

In releases prior to MATLAB Version 9.4 (R2018a), when a class overloaded subsref and numArgumentsFromSubscript and a user called an indexed expression of the form *object.method()*, MATLAB called subsref with 0 outputs without calling numArgumentsFromSubscript.

If instead the user called *object.method* (with no parentheses), then MATLAB treated this expression as a dot-reference and called numArgumentsFromSubscript. If an overloaded numArgumentsFromSubscript called the built-in version of the function, MATLAB returned 1 for the *object.method* expression. This caused errors when calling methods that return no outputs, and created differences in behavior between *object.method()* and *object.method*.

With MATLAB Version 9.4 (R2018a), when a method is called through a dot-indexing expression on the right-hand side, the built-in numArgumentsFromSubscript returns 0. Because of this change, MATLAB calls subsref with zero outputs in both the *object.method()* and *object.method* cases.

## Version History

Changed behavior can occur for cases in which classes overload numArgumentsFromSubscript and the method calls the built-in function for indexing expressions that end in dot-method (*object.method*) references.

## Classes: Concatenate matlab.lang.OnOffSwitchState enumeration members with nonmember char and string

With the release of MATLAB Version 9.4 (R2018a), the matlab.lang.OnOffSwitchState class supports special concatenation behavior to enable the formation of text expressions by concatenating enumeration members with character vectors or strings. For more information, see matlab.lang.OnOffSwitchState.

## Version History

In releases prior to MATLAB Version 9.4 (R2018a), arrays containing matlab.lang.OnOffSwitchState enumeration members and character vector or string elements required that all array elements be convertible to enumeration members (that is, on, off, true, false, 1, or 0). The resulting array was of type matlab.lang.OnOffSwitchState. With the release of MATLAB Version 9.4 (R2018a), concatenation rules enable concatenation with character vectors or strings that do not map to enumeration members. For more information, see Concatenation Rules for OnOffSwitchState.

## Python Version 3.4: Support discontinued

Support for Python version 3.4 is discontinued.

## Version History

To ensure continued support for your applications, upgrade to a supported version of Python—version 3.5 or 3.6.

## Source Control Integration: View changes, save revisions, and manage repository locks

When you are using SVN source control, the SVN file revisions history now shows a tree view of change set files that has a context menu. You can select a particular file revision and view changes or save revisions. For more information, see [Review Changes in Source Control](#).

You can now monitor and break SVN locks. The **SVN Repository Locks** dialog box supports SVN locking workflows within teams:

- Determine who has a lock
- Break locks
- Group locks by user or file

For details, see [Get SVN File Locks](#).

## MATLAB Engine API for C++: Set and get a property value on an object in an object array

With release R2018a, C++ Engine applications can call the `matlab::engine::MATLABEngine` `getProperty`, `getPropertyAsync`, `setProperty`, and `setPropertyAsync` member functions with object array inputs and pass an array index to access scalar objects within the array. In previous releases, C++ Engine applications could access object properties only from scalar object variables. For more information, see [Pass Variables from MATLAB to C++](#).

## MATLAB Data API: Applications built with R2018a API do not run in MATLAB R2017b

Applications using the MATLAB Data API built in R2018a are not supported in MATLAB R2017b.

## Version History

Applications built in MATLAB R2018a run only in MATLAB R2018a. For more information, see [Version Compatibility](#).

## MEX Functions: Build C MEX Files with Interleaved Complex API

As of MATLAB Version 9.4 (R2018a), MATLAB uses an interleaved storage representation of complex numbers. The term Interleaved complex refers to this representation, where the real and imaginary



parts are stored together. For more information, see [MATLAB Support for Interleaved Complex API in C MEX Functions](#).

This change does not affect the MATLAB language. You can continue to use the functionality described in [Complex Numbers](#) without any modification of your functions and scripts.

## Version History

If you build C MEX functions, C/C++ MEX S-functions, or standalone MATLAB engine and MAT-file applications, then you should review the [Do I Need to Upgrade My MEX Files to Use Interleaved Complex API?](#) topic. MATLAB does not support the interleaved complex API for Fortran functions.

The functionality for `mxGetPr (C)`, `mxSetPr (C)`, `mxGetPi (C)`, `mxSetPi (C)`, `mxGetData (C)`, `mxSetData (C)`, `mxGetImagData (C)`, `mxSetImagData (C)`, and `mxGetElementSize (C)` has changed. For more information, see “Functionality being removed or changed” on page 9-43.

## MEX Functions: Release-specific build options

The `mex` command has new build options, `-R2017b` and `-R2018a`, which link with release-specific versions of the C Matrix API.

- `-R2017b` — Default option. This option is equivalent to the command:  

```
mex mymex.c -largeArrayDims -DMEX_DOUBLE_HANDLE
```
- `-R2018a` — Uses the interleaved complex API, which includes the typed data access functions. For more information, see [MATLAB Support for Interleaved Complex API in C MEX Functions](#).

## Version Embedded in MEX Files

The `mex` command embeds a MEX version number in MEX files built by MATLAB R2016b and later. This number identifies the version of the Matrix API that the MEX function expects to link against at runtime.

## Version History

If you do not use the `mex` command to build your MEX files, then you must update the commands you use to build MEX files. For more information, see <https://www.mathworks.com/matlabcentral/answers/377799-compiling-mex-files-without-the-mex-command>.

## Perl 5.26.1: MATLAB support

MATLAB ships with Perl version 5.26.1.

- See [www.perl.org](http://www.perl.org) for a standard distribution of perl, perl source, and information about using perl.
- See <https://metacpan.org/pod/HTML::Parser> for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See <https://metacpan.org/pod/HTML::Tagset> for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

## Version History

If you use the `perl` command on Windows platforms, see [www.perl.org](http://www.perl.org) for information about using this version of the Perl programming language.

## System objects: Create System Objects in MATLAB

System objects are a specialized kind of MATLAB object that allow you to easily implement and simulate dynamic systems. You can use predefined System objects shipped with many System Toolboxes. You can also create your own System objects in the MATLAB editor. For more information, see [Define Basic System Objects](#).

When you create System objects in R2018a, by default, users of that System object can change characteristics of inputs, discrete states, tunable properties from call to call. New and updated methods allow you to restrict these characteristics:

Characteristic That Can Change	Methods to Restrict Characteristic
number of inputs	<code>getNumInputsImpl</code>
number of outputs	<code>getNumOutputsImpl</code>
input size	<code>isInputSizeMutableImpl</code>
input data-type	<code>isInputDataTypeMutableImpl</code>
input complexity	<code>isInputComplexityMutableImpl</code>
discrete state data-types	<code>isDiscreteStateSpecificationMutableImpl</code>
tunable property data-types	<code>isTunablePropertyDataTypeMutableImpl</code>

For more information, see [System Objects](#).

## Version History

If you want to retain strict rules for inputs, tunable properties, and discrete states, use the `sysobjupdate` function to update code in existing System objects. For more information, type `help sysobjupdate` at the MATLAB command line.

The `isInputSizeLocked` method will be removed in a future release. Use `isInputSizeMutableImpl` instead.

## System object support for strings

System objects accept strings as inputs for text input and property values.

When authoring a System object, you can use strings to define a `StringSet` property. However, the default value of a `StringSet` property must be defined as a character vector.

## .NET: Supports string data type

When calling a .NET method or function, MATLAB converts string scalar arguments to a .NET `System.String` object and string array arguments to `System.String[]`. For more information, see [Pass Data to .NET Objects](#).

The MATLAB `string` function converts `System.String` scalar arguments to a `string` scalar. The function converts `String.String[]`, `String.String[, ]`, and so on, to MATLAB string arrays with the same dimensions and sizes. Conversion of jagged arrays, for example `String.String[][]`, is not supported. For more information, see [Handle Data Returned from .NET Objects](#).

## Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	GNU® gcc and gfortran version 6.x. Version 6.3 recommended.	Linux
Discontinued	GNU gcc and gfortran version 4.9.x	Linux
Discontinued	Apple Xcode 7.x	macOS
Discontinued	Intel C++ Composer XE 2013	Windows
Discontinued	Intel Visual Fortran Composer XE 2013	Windows
Discontinued	Intel Fortran Composer XE 2013	macOS
To be phased out	Visual C++ 2013 Professional	Windows

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see [Supported and Compatible Compilers](#).

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Programmatic dependence on specific diagnostic subclass from <code>getDiagnosticFor</code> method of <code>constraint</code> and <code>tolerances</code>	Warns	Not applicable	<p>Rework code that relies on properties or methods specific to <code>matlab.unittest.diagnostic.ConstraintDiagnostic</code> instances returned from the <code>getDiagnosticFor</code> method of <code>matlab.unittest.constraints</code> classes.</p> <p>In a future release, diagnostics returned from <code>constraint</code> and <code>tolerance</code> classes in the <code>matlab.unittest.constraints</code> package will be subclasses of <code>matlab.unittest.diagnostics.Diagnostic</code> and might not be instances of <code>matlab.unittest.diagnostic.ConstraintDiagnostic</code>.</p>
Negation of <code>matlab.unittest.constraints.ReturnsTrue</code> constraint ( <code>~ReturnsTrue</code> )	Errors	Not applicable	Change the logic for tests that rely on negating the <code>ReturnsTrue</code> constraint.

<b>Functionality</b>	<b>Result</b>	<b>Use This Instead</b>	<b>Compatibility Considerations</b>
mxGetPi and mxSetPi in C MEX functions built with the interleaved complex API (mex option -R2018a)	Errors	Use mxGetComplexDoubles instead of mxGetPr and mxGetPi.	MATLAB Support for Interleaved Complex API in C MEX Functions
mxGetImagData and mxSetImagData in C MEX functions built with the interleaved complex API (mex option -R2018a)	Errors	Typed Data Access functions	MATLAB Support for Interleaved Complex API in C MEX Functions
mxGetPr and mxSetPr in C MEX functions built with the interleaved complex API (mex option -R2018a)	Still runs when called for real mxArray's.  Errors when called for complex mxArray's.	Typed Data Access functions	MATLAB Support for Interleaved Complex API in C MEX Functions
mxGetData and mxSetData in C MEX functions built with the interleaved complex API (mex option -R2018a)	Still runs	Typed Data Access functions	Use mxGetData (C) and mxSetData (C) for nonnumeric arrays only.
mxGetElementSize in C MEX functions built with the interleaved complex API (mex option -R2018a)	Still runs	Not applicable	mxGetElementSize (C)
mexSetTrapFlag in C and Fortran Matrix API	Errors	mexCallMATLABWithTrap	mexCallMATLABWithTrap lets you catch, or trap, errors.

# R2017b

---

**Version: 9.3**

**New Features**

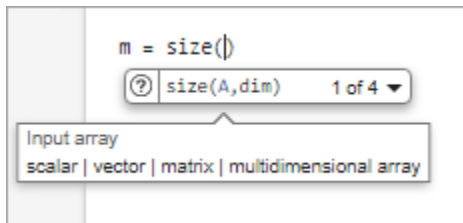
**Bug Fixes**

**Version History**

## Desktop

### Live Editor: Write MATLAB commands with automated, contextual hints for arguments, property values, and alternative syntaxes

When writing commands, MATLAB automatically displays contextual hints for arguments, property values, and alternative syntaxes. For example, if you want to use the `size` function, MATLAB displays the syntax information to help you write the command.



MATLAB also automatically suggests and completes the names of functions, models, MATLAB objects, files, folders, variables, structures, graphics properties, parameters, and options. For more information, see Automatic Code Suggestions and Completions.


### Live Editor: Export live scripts to LaTeX format

To export your live script to LaTeX format, on the **Live Editor** tab, select **Save > Export to LaTeX**. MATLAB creates a separate Extensible Stylesheet Language (XSL) file in the same folder as the output document, if one does not exist already. XSL files give you more control over the appearance of the output document. For more details, see Share Live Scripts.

### Live Editor: Display high-resolution plots in PDF output

When saving a live script as a PDF, MATLAB now includes high-resolution plots in the PDF file.

### Live Editor: Horizontally align text, equations, and images

You can horizontally align text, equations, and images in live scripts. On the **Live Editor** tab, in the **Text** section, select left , center , or right .

For more information, see Format Live Scripts.

### Live Editor: Automatically match delimiters and wrap comments while editing code

MATLAB autocompletes parentheses and quotes when entering code in the Live Editor. For example, if you type `r = rand(`, MATLAB automatically adds the closing parentheses to the statement (`r = rand()`). The Live Editor also autocompletes any comments, character vectors, and strings that are split across two lines.

For more information, see Run Code.

## Live Editor: View and scroll through table data, including variable and row names

Use scroll bars to explore data in tables and timetables in live scripts. View the variable and row names included in the tables.

T = 100x10 table ?

	LastName	Gender	Age	Location	Height	Weight	Smoker	Systolic	Diastolic	Self
23	'Lewis'	'Female'	41	'VA Hospital'	62	137	0	114	88	'Fai
24	'Lee'	'Female'	44	'County Ge...	66	146	1	128	90	'Fai
25	'Walker'	'Female'	28	'County Ge...	65	123	1	129	96	'Go
26	'Hall'	'Male'	25	'VA Hospital'	70	189	0	114	77	'Poc
27	'Allen'	'Female'	39	'VA Hospital'	63	143	0	113	80	'Exc
28	'Young'	'Female'	25	'County Ge...	63	114	0	125	76	'Go
29	'Hernandez'	'Male'	36	'County Ge...	68	166	0	120	83	'Poc
30	'King'	'Male'	30	'County Ge...	67	186	1	127	89	'Exc
31	'Wright'	'Female'	45	'VA Hospital'	70	126	1	134	92	'Exc
32										

## Live Editor: Check code for errors and warnings using the message bar and message indicator

Determine whether a live script contains an error or warning using the message indicator. Navigate through errors and warnings using the message bar. For more information, see [Check Code for Errors and Warnings](#).

## Documentation: Use the Live Editor in a web browser to open, edit, and run MATLAB online documentation examples

Open, edit and run live script examples directly from the MATLAB online documentation using the Live Editor in a web browser. To open an example, click the **Try this Example** button to the right of the example and select **Try it in your browser**.


## MATLAB Drive: Store, access, and manage your files from anywhere

If you have MATLAB Drive Connector installed on your system, you can access files and folders in your drive from MATLAB using the Current Folder browser.

To view your recent activity, in the Current Folder browser, right-click any MATLAB Drive file or folder and select **MATLAB Drive > View Recent Activity...** To open MATLAB Drive online, right-click any MATLAB Drive file or folder and select **MATLAB Drive > Go to MATLAB Drive Online...**

For more information, see [Manage Files and Folders](#) or the [MATLAB Drive documentation](#).

## Add-On Manager: Customize your MATLAB environment by enabling and disabling add-ons

Enable and disable add-ons in the Add-On Manager to customize your MATLAB environment. To enable an add-on in the Add-On Manager, click the  button to the right of the add-on and select **Enabled**. This option is only available for apps, toolboxes, functions, collections and Simulink models.

You also can enable or disable the add-on using the `matlab.addons.enableAddon` and `matlab.addons.disableAddon` functions.

## Add-On Manager: Find installed add-ons faster using sort and search

Use sort and search in the Add-On Manager to find installed add-ons more efficiently.

## Toolbox Packaging: Create a Getting Started Guide for your toolbox from a Live Script template

You can create a Getting Started Guide when you package your toolbox. Users of your toolbox can view the Getting Started Guide through the Options menu for the toolbox in the Add-On Manager. After you select your toolbox folder, the option to create a Getting Started Guide appears in the **Examples, Apps, and Documentation** section of the Package a Toolbox dialog box. For more information, see [Create and Share Toolboxes](#).

## Toolbox Packaging: Share your toolbox on File Exchange directly when you package it

When you are creating your toolbox, you can package the toolbox and share it on MATLAB Central File Exchange. Select **Package and Share** from the **Package** menu at the top of the Package a Toolbox dialog box. This option opens a web page for your toolbox submission to File Exchange submission. MATLAB populates the File Exchange submission form with information from the Package a Toolbox dialog box. Review and submit the form to share your toolbox on File Exchange. For more information, see [Create and Share Toolboxes](#).

## Command Window: View updated display for cell arrays

When displaying cell array outputs, the Command Window now displays curly braces around each cell. The curly braces show the delimiters of the cell and help unify the display of the cell array.

Use curly braces to access the contents of a cell. For example, this code creates a 2-by-3 cell array of text and numeric data, and then displays the contents of the last cell in the cell array.

```
C = {'one', 'two', 'three'; 1, 2, 3}
last = C{2,3}
```

```
C =
    2×3 cell array
    {'one'}    {'two'}    {'three'}
    {[ 1]}    {[ 2]}    {[ 3]}
```

```
last =
    3
```



Additionally, if a cell contains a numeric empty value, then the Command Window displays the class and size of the cell.

For more information, see [Create Cell Array](#).

## Language and Programming

### **Code Compatibility Report: Generate a report that helps the updating of code to a newer MATLAB release**

You can generate a report of potential compatibility issues in your code using the `codeCompatibilityReport` function. For example, the report contains information about the use of discouraged or removed functions in your code and the occurrence of invalid syntaxes. After you upgrade to a newer version of MATLAB, you can use this report to identify potential compatibility issues in your existing code.

Alternatively, you can create a `CodeCompatibilityAnalysis` object to save results using the `analyzeCodeCompatibility` function.

### **isStringScalar Function: Determine whether input is a string array with one element**

To determine whether the input argument is a string array that has only one element, use the `isStringScalar` function.

### **convertStringsToChars and convertCharsToStrings Functions: Enable your code to accept all text types as inputs without otherwise altering your code**


To make your existing code accept strings as input arguments, you can use the `convertStringsToChars` function on the entire input argument list. `convertStringsToChars` converts input string arrays to character vectors or cell arrays of character vectors while returning all the other input arguments unaltered. If you add `convertStringsToChars` to the beginning of your code, then you do not need to make any other changes to accept strings as inputs.

Similarly, `convertCharsToStrings` converts input character vectors or cell arrays of character vectors to string arrays while returning the other input arguments unaltered. If you have code that works with strings, then you can add `convertCharsToStrings` to the beginning so that it also accepts character arrays as inputs.

### **arrayfun, cellfun, and structfun Functions: Return object arrays as output arguments**

The `arrayfun`, `cellfun`, and `structfun` functions can return object arrays of any data type, so long as the objects can be concatenated.

### **Scripts: Run sections in scripts containing local functions**

You can run an individual section in a script that contains local functions. To run a section, on the **Editor** or **Live Editor** tab, click  **Run Section**.

---

**Note** You must fix all syntax errors in the script before running an individual section.

---

## isfile and isfolder Functions: Determine if input is a file or a folder

Use the `isfile` and `isfolder` functions to determine if an input is a file or a folder located on the specified path or in the current folder.

For example, this code creates a folder, and then it uses the `isfile` and `isfolder` functions to check whether the input is a file or folder:

```
mkdir myfolder;
result1 = isfile('myfolder')
result2 = isfolder('myfolder')

result1 =
    logical
     0

result2 =
    logical
     1
```

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
The <code>cellstr</code> function converts missing values in an input string array to 0-by-0 character arrays in the output cell array.	Still runs	Not Applicable	Starting in R2017b, <code>cellstr</code> converts missing values in string arrays to 0-by-0 character arrays. In R2016b and R2017a, <code>cellstr</code> throws an error if the input argument is a string array that contains missing values.
<code>isdir</code>	Still runs	<code>isfolder</code>	Replace all instances of <code>isdir</code> with <code>isfolder</code> .  <code>isdir</code> searches for folders on the search path, which can lead to unexpected results. <code>isfolder</code> searches for folders only on the specified path or in the current folder.
On Windows, <code>tempdir</code> checks first for the existence of <code>TMP</code> , and then for <code>TEMP</code> .	Still runs	Not Applicable	On Windows, the <code>tempdir</code> function now checks first for the existence of the <code>TMP</code> environment variable, and then it checks for the <code>TEMP</code> environment variable. It uses the first path found. Previously, <code>tempdir</code> checked first for the existence of <code>TEMP</code> .
On Linux and Mac, <code>tempdir</code> function uses <code>TMPDIR</code> .	Still runs	Not Applicable	On Linux and Mac, the <code>tempdir</code> function now uses the <code>TMPDIR</code> environment variable, if it is defined.

Functionality	Result	Use This Instead	Compatibility Considerations
Using a MATLAB identifier to call a function, and then using it as a variable name inside the same function	Warns	Choose different MATLAB identifiers for local or imported functions and variable names.	<p>In a future release, using the same identifier for a local or imported function, and then using it for a variable name will result in an error. For example, in a future release, both of the following functions will result in an error.</p> <pre>function myFunc()     x = containers.Map;     containers = 1; end function myFunc2()     import java.lang.*     % includes java.lang.String     s = String('Hello');     String = 1; end</pre>

Functionality	Result	Use This Instead	Compatibility Considerations
Indexing into an implicitly defined variable in a function	Still runs	Explicitly define a variable before indexing in to it.	<p>Currently, an identifier without an explicit declaration is classified as a variable when it is indexed with the colon, end, or curly braces. For example, <code>X(a,b,:)</code>, <code>X(end)</code>, or <code>X{a}</code>. In a future release, if a function of the same name exists on the path, the variable will be classified as a function.</p> <p>Consider the following code:</p> <pre>function myfunc()     load data.mat;     disp(x(:)) end</pre> <p>If you intend to use <code>x</code> as a variable from <code>data.mat</code> instead of a function, explicitly define it.</p> <pre>function myfunc()     load data.mat x;     disp(x(:)) end</pre> <p>Similarly, to use <code>x</code> as a variable obtained from a script, define the variable before invoking the script. This behavior also applies if the variable is implicitly introduced by any of the following functions: <code>sim</code>, <code>eval</code>, <code>evalc</code>, and <code>assignin</code>. For example, if <code>myscript.m</code> defines <code>x</code>, define <code>x</code> before calling <code>myscript</code>.</p> <pre>function myfunc2()     x = [];     myscript;     disp(x(:)) end</pre>

Functionality	Result	Use This Instead	Compatibility Considerations
Sharing uninitialized variables between a nested function and the parent function	Still runs	Explicitly define shared variables in the parent function before calling the nested function.	<p>Consider the following code.</p> <pre data-bbox="914 352 1214 552">function myfunc()     nested();     x;     function nested()         x = 1;     end end</pre> <p>Currently, if <code>x</code> is a function on the path, MATLAB classifies it as a function in <code>myfunc</code> and a variable in <code>nested</code>. Otherwise, MATLAB classifies it as a variable shared between <code>nested</code> and <code>myfunc</code>.</p> <p>To avoid potential function behavior differences depending on the state of your MATLAB path, explicitly define shared variables in the parent function.</p> <pre data-bbox="914 915 1214 1140">function myfunc()     x = 0;     nested();     x;     function nested()         x = 1;     end end</pre>

Functionality	Result	Use This Instead	Compatibility Considerations
Nested functions in MATLAB do not inherit import statements from the parent function.	Still runs	Not Applicable	<p>The <code>plot</code> statement in the following nested function calls the <code>plot</code> function on the path, not the <code>pkg.plot</code> function.</p> <p>In the future, MATLAB will call the <code>plot</code> function in <code>pkg.plot</code>.</p> <pre>function myfunc import pkg.plot     function nested         plot     end end</pre> <p>Similarly, in the following example, even if the <code>plot</code> function is defined in <code>pkg.*</code>, MATLAB calls the function on the path. However, in the future, MATLAB will call the function in imported package <code>pkg.*</code>.</p> <pre>function myfunc import pkg.*     function nested         plot     end end</pre>
<code>ans</code> in nested functions	Still runs	Not Applicable	<p>Nested functions now access and modify the <code>ans</code> variable that is defined in the parent function. In future releases, nested functions will have access to the <code>ans</code> variable even if the variable is declared implicitly by not suppressing the output. For example in future releases, in this code, <code>ans</code> returns 1.</p> <pre>function foo     a=1;     a % Assigns "ans" to 1.     nested()     function nested         ans % will be 1     end end</pre>
Using <code>ans</code> as a function name	Warns	Not Applicable	Rename all functions with the name <code>ans</code> . In a future release, using <code>ans</code> as a function name issues an error.
Declaring a variable as global after referencing it	Warns	Declare variables as global before using them.	Currently, if you declare a variable as global after referencing it, MATLAB issues a warning and the variable becomes a global variable. In a future release, this will result in an error.

<b>Functionality</b>	<b>Result</b>	<b>Use This Instead</b>	<b>Compatibility Considerations</b>
MATLAB disables folder change detection for all of <i>matlabroot</i>	Still runs	Not applicable	MATLAB now disables folder change detection for all of the files and folders in the <i>matlabroot</i> folder. Previously, MATLAB only disabled folder change detection for the files and folders in <i>matlabroot/Toolbox</i> . For more information, see Toolbox Path Caching in MATLAB.



## Mathematics

### **decomposition Object: Solve linear systems repeatedly with improved performance**

`decomposition` objects efficiently store a matrix decomposition for repeated use in solving linear systems  $Ax = b$  or  $xA = b$ . These objects enable you to leverage the performance benefits of precomputing the matrix decomposition, but they *do not* require knowledge of how to use the matrix factors.

### **lsqminnorm Function: Find minimum-norm solution of underdetermined linear system**

For underdetermined least-squares problems  $A*x \approx b$  where several equivalent least-squares solutions exist, `x = lsqminnorm(A,b)` returns the least-squares solution `x` that minimizes `norm(x)` among all vectors that minimize `norm(A*x-b)`.

Previously, this functionality was available in a limited fashion using `x = pinv(A)*b`. However, `lsqminnorm` is faster since it does not need to compute the pseudoinverse `pinv(A)`, and it also supports sparse matrices.

### **dissect Function: Reorder sparse matrix columns using nested dissection ordering**

Use the `dissect` function to generate fill reducing orderings of sparse matrices. The fill reducing ordering produced by `dissect` tends to reduce storage and computation time for sparse matrix factorizations when compared to other reordering functions such as `amd`.

### **vecnorm Function: Compute vector-wise norms of arrays**

Use `vecnorm` to compute the norm of the rows and columns of a numeric N-dimensional array. Unlike the `norm` function that computes the matrix norm, `vecnorm` treats the elements along a specified dimension as vectors and calculates the norm of each vector.

### **polyshape Object: Create, analyze, and visualize 2-D polygons**

The `polyshape` function enables you to create a polygon from a list of 2-D vertices. For example, `p = polyshape([0 0 1 1],[0 1 1 0])` creates a square with vertices (0,0), (0,1), (1,1), and (1,0).

After you create a `polyshape` object `p`, use the command `plot(p)` to visualize it. You also can compute common geometric quantities of `p`, such as its centroid and area. You can transform `p` using translation, rotation, or scaling, and you can query properties of `p` such as its number of boundaries or holes. If you are working with multiple polygons, then you can compute quantities, such as their intersection and union.

### **eigs Function: Improved algorithm and new options**

The `eigs` function has a new algorithm, more intuitive behavior, and new options to simplify usage.

- **New sigma options**

This table summarizes the new, more descriptive options for `sigma`. These changes do not require any updates to existing code since the old values continue to be supported.

Old sigma	New sigma	Which eigenvalues?
'LM'	'largestabs'	$\max(\text{abs}(d))$
'SM'	'smallestabs'	$\min(\text{abs}(d))$
'LA' (if real symmetric), 'LR' (otherwise)	'largestreal'	$\max(\text{real}(d))$
'SA' (if real symmetric), 'SR' (otherwise)	'smallestreal'	$\min(\text{real}(d))$
'BE' (if real symmetric)	'bothendsreal'	Combination of values returned by 'largestreal' and 'smallestreal'
'LI' (if complex)	'largestimag'	$\max(\text{imag}(d))$
'SI' (if complex)	'smallestimag'	$\min(\text{imag}(d))$
'LI' (if real nonsymmetric)	'bothendsimag'	Combination of values returned by 'largestimag' and 'smallestimag'

- **sigma values 'LR', 'LA', 'SR', 'SA', and 'BE' accept both symmetric and nonsymmetric matrices**

If an eigenproblem is symmetric, it has real eigenvalues. For that reason 'LR'/'LA' and 'SR'/'SA' now are considered to be equivalent by `eigs`.

In the nonsymmetric case, the 'BE' option uses the real part of each eigenvalue to determine which eigenvalues to return. The behavior is unchanged for problems that previously allowed the 'BE' option.

- **Alternative name-value pairs for options**

This table contains a one-to-one mapping of the old option structure fields to the corresponding new name-value pairs. These changes do not require any updates to existing code since the options structure continues to be supported.

Options Structure Field	Name-Value Pair	New Behaviors
issym	'IsFunctionSymmetric'	This option is used only for function handle inputs. For matrix inputs, <code>eigs</code> ignores this option.
isreal	-	<code>eigs</code> ignores this option because the new algorithm does not need to know whether the input matrix is real or complex.
tol	'Tolerance'	New default value of $1e-14$ .
maxit	'MaxIterations'	

Options Structure Field	Name-Value Pair	New Behaviors
p	'SubspaceDimension'	
v0	'StartVector'	eigs uses the same starting vector for each call so that the output is reproducible between calls.
disp	'Display'	A display value of 2 no longer returns timing information. Instead, eigs treats a value of 2 the same as a value of 1. Also, the messages shown by the 'Display' option have changed. The new messages show the residual in each iteration, instead of the Ritz values.
spdB	'IsSymmetricDefinite'	New option.
cholB	'IsCholesky'	
permB	'CholeskyPermutation'	
fail	'FailureTreatment'	New option.

## Version History

The eigs function also has some new behaviors that affect the output and might require updates to code.

- **Changes to sorting order of output**

eigs now sorts the output according to the value of sigma. For example, the command `eigs(A,k,'largestabs')` produces k eigenvalues sorted in descending order by magnitude.

Previously, the sorting order of the output produced by eigs was not guaranteed.

- **Reproducibility**

Calling eigs multiple times in succession now produces the same result. Set 'StartVector' to a random vector to change this behavior.

- **Display**

A display value of 2 no longer returns timing information. Instead, eigs treats a value of 2 the same as a value of 1. Also, the messages shown by the 'Display' option have changed. The new messages show the residual in each iteration, instead of the Ritz values.

## svds Function: Set options with name-value pairs

You can use name-value pairs to set options with svds instead of using an options structure. This table contains a one-to-one mapping of the old option structure fields to the corresponding new name-value pairs.

These changes do not require any updates to existing code since the options structure continues to be supported.

Options Structure Field	Name-Value Pair
tol	'Tolerance'
maxit	'MaxIterations'
p	'SubspaceDimension'
u0	'LeftStartVector'
v0	'RightStartVector'
fail	'FailureTreatment'
disp	'Display'

## Interpolation Functions: Method for modified Akima cubic Hermite interpolation

The `griddedInterpolant`, `interp1`, `interp2`, `interp3`, and `interpN` functions now support the interpolation method `'makima'`. This modified Akima cubic Hermite interpolation method has these properties:

- It is  $C^1$  continuous.
- It produces fewer undulations than `'spline'`, but the result is not as aggressively flattened as `'pchip'`.
- Unlike `'pchip'`, it supports N-D arrays.
- Unlike `'spline'`, it does not produce overshoots.

## convn Function: Compute convolutions on multidimensional arrays with improved performance

You can compute convolutions on multidimensional arrays faster with the `convn` function.

## subgraph and highlight Functions: Specify graph nodes with logical vector

The `subgraph` and `highlight` functions for network analysis now accept a logical vector to select nodes.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
<code>[bins,edges] = discretize(X,dur)</code> where X is a datetime or duration array	Still runs	N/A	In R2017a and later releases, <code>discretize(X,dur)</code> operating on a datetime or duration array X uses a maximum of $2^{16} = 65536$ bins. This aligns the behavior of <code>discretize</code> with that of <code>histcounts</code> and <code>histogram</code> . In releases before R2017a, <code>discretize</code> did not impose this limitation on the number of bins in the result.

## Graphics

### **geobubble Function: Create an interactive map with bubbles whose size and color vary with data values**

Create an interactive map with bubbles whose size and color vary with data values using `geobubble`.

### **wordcloud Function: Display words at different sizes based on frequency or custom size data**

Display words at different sizes based on frequency or custom size data using `wordcloud`.

### **binscatter Function: Visualize data density with dynamic bin size adjustment**

Use `binscatter` to visualize binned scatter plots and identify trends and relationships in arrays of data. `binscatter` produces a plot that is similar to a 2-D histogram (`histogram2`). However, since the `binscatter` function has custom zoom behavior that increases the resolution as you zoom in, it is better suited for visual exploration of data.

### **Tall Array Support: Visualize out-of-memory data using plot, scatter, and binscatter**

The `plot`, `scatter`, and `binscatter` functions now support tall arrays. These functions plot the data in iterations, progressively adding to the plot as more data is read. Zooming and panning is supported during the updating process before the plot is complete.

### **heatmap Function: Sort rows and columns and use custom labels in a heatmap**

You can make new types of modifications to a heatmap, such as reordering or relabeling the values along each axis. Use the new `HeatmapChart` listed in this table. If you want to reorder the values, you also can use the `sortx` and `sorty` functions.

Property	Description
XData and YData	Values associated with the rows or columns in <code>ColorData</code> .
XDisplayData and YDisplayData	Order of values along the x-axis or y-axis. Use these properties to view, rearrange, or show only a subset of the values.
XDisplayLabels and YDisplayLabels	Labels for x-axis or y-axis values. Use these properties to view or relabel the values.
XLimits and YLimits	First and last value displayed along the x-axis or y-axis.

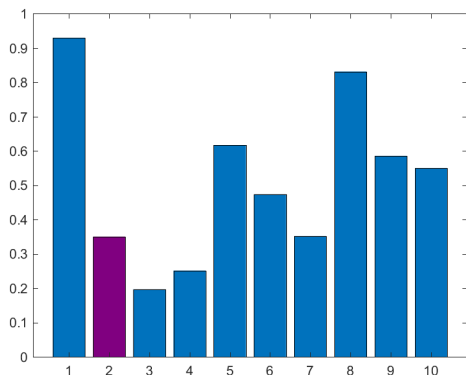
Property	Description
ColorDisplayData	Values as they appear in the heatmap, sorted based on XDisplayData and YDisplayData. This is a read-only property.

For an example, see [Create Heatmap from Tabular Data](#).

## bar Function: Control individual bar colors

Specify a different color for each bar using the new CData property for Bar objects. For example, create a bar chart and change the color of the second bar.

```
b = bar(rand(10,1));
b.FaceColor = 'flat';
b.CData(2,:) = [.5 0 .5];
```

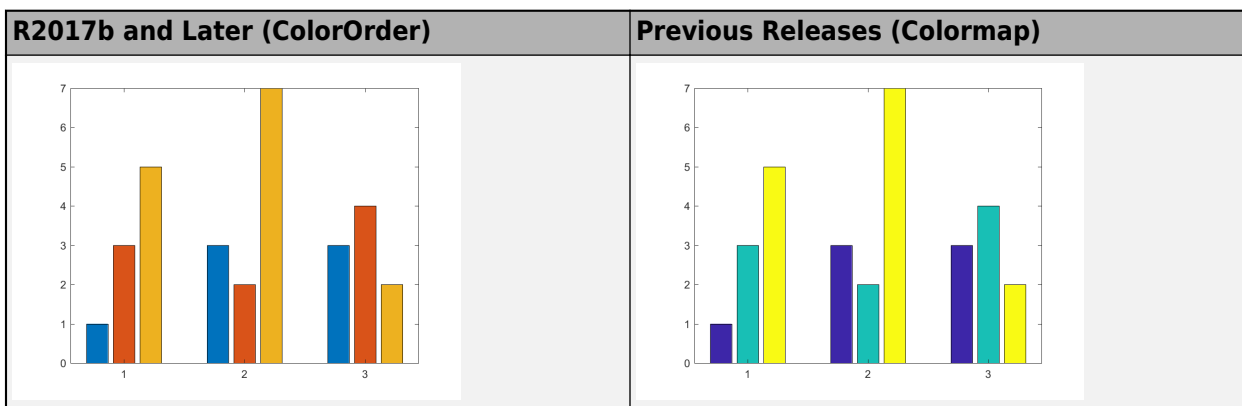


For more information, see [Bar Properties](#).

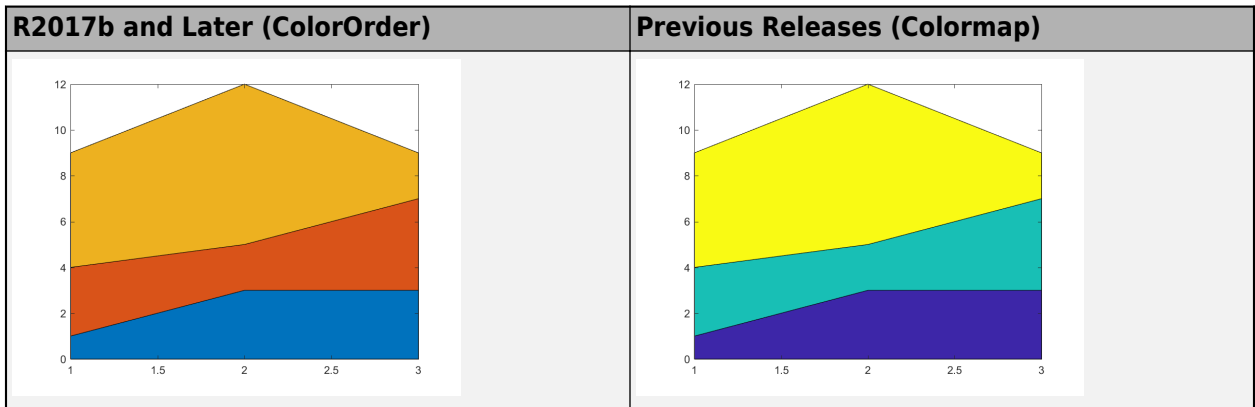
## Chart Colors: Create bar and area charts with new default colors

Similar to line charts, bar charts and area charts now use colors from the ColorOrder property of the Axes object. Previously, the charts used colors from the colormap.

This table shows the color difference for bar charts.



This table shows the color difference for area charts.



## Version History

Bar charts and area charts use different colors than in previous releases.

- To create a bar chart that uses colormap colors, set the `FaceColor` property to `'flat'`. Then set the `CData` property for each `Bar` object to an integer. For example:

```
y = [1 3 5; 3 2 7; 3 4 2];
b = bar(y, 'FaceColor', 'flat');
for k = 1:size(y,2)
    b(k).CData = k;
end
```

- To create an area chart that uses colormap colors, set the `FaceColor` property to `'flat'`. For example:

```
y = [1 3 5; 3 2 7; 3 4 2];
area(y, 'FaceColor', 'flat')
```

## Axes Object: Specify the target axes for more functions

These functions now support an Axes object as the first input argument: `camlight`, `hidden`, `ishold`, `lightangle`, `lighting`, and `material`.



## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Default colors for bar and area charts	Still runs	Not applicable	Bar and area charts have new default colors. These charts use colors from the color order instead of colormap colors. For more information, see “Chart Colors: Create bar and area charts with new default colors” on page 10-19.
Default axis limits for bar charts, histograms, and graph plots	Still runs	Not applicable	Bar charts, histograms, and graph plots might use slightly different limits than in previous releases. These charts automatically choose limits that center the data. If you want to specify different limits, use the <code>xlim</code> and <code>ylim</code> functions.
<code>barh</code> function with the <code>Horizontal</code> parameter	Warns	<code>bar</code> function	Remove instances of code that uses the <code>barh</code> function with the <code>Horizontal</code> parameter. To create a vertical bar chart, use the <code>bar</code> function. To create a horizontal bar chart, use the <code>barh</code> function.
<code>bar3</code> function with a line style or marker symbol specification	Errors	Not applicable	Remove instances of code that passes a line style or marker symbol to the <code>bar3</code> function, such as <code>bar3(y,z,'r:*')</code> . In previous releases, the line style and marker symbol are ignored. Instead, specify only the color, such as <code>bar3(y,z,'r')</code> .

Functionality	Result	Use This Instead	Compatibility Considerations
legend function	Still runs	Not applicable	<p>The legend function now always creates a legend associated with the current or specified axes. If the axes do not contain any data, then the legend function creates an empty legend. If axes do not exist, then the legend function creates axes with an empty legend.</p> <p>In previous releases, the legend function does not always create a legend. For example, it does not create a legend if there are no axes or no data in the axes. For these cases, the function returns a warning message instead. Also, the <code>lgd = legend</code> syntax (with no input arguments) returns an existing legend, but does not create one if a legend does not already exist.</p> <p>To query for the existence of a legend associated with a particular axes without creating a new one, use the Legend property of the Axes object. For example:</p> <pre>ax = gca; lgd = ax.Legend</pre>
subplot function when setting the OuterPosition or ActivePositionProperty property as name-value pair arguments	Warns	Position parameter	Remove instances of code that sets the OuterPosition or ActivePositionProperty property as name-value pair arguments to the subplot function. If you want to specify the subplot position, use the Position property.

Functionality	Result	Use This Instead	Compatibility Considerations
subplot function when specifying the subplot location using both the grid position and the Position parameter	Warns	Either a grid position or the Position parameter (not both)	You cannot specify the subplot location using both a grid position and the Position parameter, such as subplot(m,n,p,'Position',[x y w h]). Instead, use either the grid position or the Position parameter, such as subplot(m,n,p) or subplot('Position',[x y w h]).
Behavior of charting functions when the NextPlot property of the Axes object is set to 'replacechildren'	Still runs	Not applicable	Charting functions no longer change most axes properties when the NextPlot property of the Axes object is set to 'replacechildren'.

## Data Import and Export

### Custom Datastore: Build a customized datastore

Build your own datastore for custom or proprietary data (stored in files or a data stream) using the custom datastore framework. Then, use this custom datastore to bring your data into MATLAB and leverage the MATLAB Big Data capabilities such as `tall`, MapReduce, and Hadoop. For more information, see [Develop Custom Datastore](#).

### datastore Function: Work with data stored in Windows Azure Blob Storage

You can use data stored on Windows Azure to create a datastore. Windows Azure Blob Storage (WABS) is an online file storage web service offered by Microsoft. For more information, see [Introduction to Windows Azure](#).

To use Windows Azure with datastore, see [Windows Azure Blob Storage](#).

### datastore Function: Access Hadoop HDFS data more easily

You can access Hadoop data more easily by using the `hdfs:///` syntax:

```
ds = datastore('hdfs:///path_to_file/file.ext')
```

For example, this command creates a datastore for file `myData.txt` in folder `data`.

```
ds = datastore('hdfs:///data/myData.txt')
```

When you specify location with this syntax, the `datastore` function uses the Hadoop HDFS host name associated with the Hadoop installation that MATLAB is paired with.

### FileDatastore Object: Create uniform output from datastore

For vertically concatenated data, you can design your `ReadFcn` to return a uniform output, in the form of a MATLAB table.

For example, create a `FileDastore` object and check the value of its new `UniformRead` property. This property indicates whether multiple reads of `FileDatastore` return uniform data that can be vertically concatenated. The `UniformRead` property is set to either `true` or `false`.

- If set to `false` (default), then the `readall` method returns a cell array of data where each call to the `read` method adds new entries into the cell array.
- If set to `true`, then `readall` method returns a table. Therefore, your `ReadFcn` must return vertically concatenatable data else the `readall` method returns an error.

### HDF5 Functions: Create datasets, groups, attributes, links, and named datatypes using non-ASCII characters

HDF5 functions support use of non-ASCII characters for HDF5 datasets, groups, attributes, links, and named datatypes. To create and access non-ASCII data, set the `'TextEncoding'` name-value pair to `'UTF-8'`. These high-level and low-level functions support the `'TextEncoding'` name-value pair:

### High-Level Functions

- `h5create`
- `h5writeatt`
- `h5info`
- `h5disp`

### Low-Level Functions

- `H5A.get_name`
- `H5I.get_name`
- `H5L.get_name_by_idx`
- `H5L.get_val`
- `H5R.get_name`

For more information, see [Working with Non-ASCII Characters in HDF5 Files](#).

### Web services: Skip server name verification in certificates

To disable certificate server name verification in cases where the server's certificate does not match the URI used to access it, set the `VerifyServerName` property to `false` in `matlab.net.http.HTTPOptions`.

### jsonencode Function: Encode NaN and Inf as null

The `jsonencode` function encodes MATLAB NaN and Inf values in JSON as null.

### Version History

To continue encoding NaN as 'NaN', call `jsonencode` with the `'ConvertInfAndNaN'` option set to `false`.

```
jsonencode(NaN, 'ConvertInfAndNaN', false)
```

```
ans =
```

```
    'NaN'
```

## Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
ncwrite	Still runs	netcdf.putVar	<p>The ncwrite function applies attribute conventions <code>_FillValue</code>, <code>scale_factor</code>, and <code>add_offset</code> when writing data of all data types.</p> <p>If you do not want to apply attribute conventions when you write data to a netCDF variable, use <code>netcdf.putVar</code>.</p>

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<p>Using load statements inside a function to load variables from file.</p> <p>For example:</p> <pre>function loadVar()     ...     load data.mat x;     disp(x)     ... end</pre>	Warns	none	<p>Currently, if you use a load statement in a function to load a variable that does not exist, then MATLAB behaves as follows:</p> <ol style="list-style-type: none"> <li>1 Issues a warning 'Variable not found'.</li> <li>2 Interprets any subsequent references to that variable, as calls to a path function (with the same name as the variable) if one exists.</li> </ol> <p>For example, consider a file <code>data.mat</code> that has only two variables, <code>a</code> and <code>b</code>. Write a function to load a variable <code>pi</code> from <code>data.mat</code> and include a subsequent call to <code>pi</code>.</p> <pre>function loadVar()     load data.mat pi; % pi does not exist in d     pi; end</pre> <p>Calling <code>loadVar</code> results in a warning. Then, <code>pi</code> in the next line gets resolved to <code>pi</code> on the path.</p> <pre>Warning: Variable 'pi' not found. &gt; In loadVar (line 2)</pre> <pre>ans =     3.1416</pre> <p>In future releases, if you use a load statement in a function to load a variable that does not exist, then MATLAB:</p> <ol style="list-style-type: none"> <li>1 Issues the warning 'Variable not found'.</li> <li>2 Interprets any subsequent references to that variable, as calls to a variable. Therefore, MATLAB returns the error 'Undefined function or variable'.</li> </ol> <p>Attempting to load <code>pi</code> from <code>data.mat</code> and a subsequent call to <code>pi</code>, results in a warning and an error.</p>

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
			Warning: Variable 'pi' not found. > In loadVar (line 2) Undefined function or variable 'pi'.  Error in loadVar (line 3) pi



## Data Analysis

### ischange Function: Detect abrupt changes in data

You can use the `ischange` function to find local changes in the mean, variance, or linear slope and intercept of array and table data.

### islocalmin and islocalmax Functions: Detect local minima and maxima in data

To find local minima and maxima in table or array data, use the `islocalmin` and `islocalmax` functions.

### rescale Function: Scale data to a specified range

You can scale data in an array to a specific range using the `rescale` function. For example, `rescale(A,1,10)` scales the entries of an array `A` to the interval `[1,10]`.

### tall Arrays: Operate on tall arrays with more functions, including fillmissing, filter, median, polyfit, and synchronize

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For information on usage and limitations, type `help tall/functionName`. For example, `help tall/filter`.

<code>binscatter</code>	<code>plot</code>	<code>scatter</code>
<code>bounds</code>	<code>polyfit</code>	<code>split</code> (for tall arrays of strings)
<code>fillmissing</code>	<code>polyval</code>	<code>summary</code> (for tall timetables)
<code>filter</code>	<code>retime</code> (for tall timetables)	<code>synchronize</code> (for tall timetables)
<code>median</code>	<code>rmmissing</code>	<code>vertcat</code>

### tall Array Indexing: Use subscripted assignment with tall arrays

The subscripted assignment statement `A(m,n,p,...) = B` adds support for these situations:

- The right side of the assignment statement `B` can be a tall array derived from `A`. For example, with a 3-D tall array `A = tall(ones(3,3,3))`, you can make assignments such as `A(:,:,1) = mean(A,3)`.
- The subscripts `n,p,...` can exceed the dimensions of `A`, which enables you to grow the array with subscripted assignment for any dimension other than the first (tall) dimension. For example, with a 2-D tall matrix `A = tall(rand(1000,3))`, you can turn `A` into a 3-D matrix with `A(:,:,2) = 2*A(:,:,1)`.

## **tallrng Function: Control random number generator used by tall arrays**

Use `tallrng` in a manner similar to `rng` to control the random number generator used in certain tall array calculations.

## **timetable Data Container: Specify whether each variable in a timetable contains continuous or discrete data using the VariableContinuity property**

To specify whether `timetable` variables represent continuous data, you can use the `VariableContinuity` property. If you specify `VariableContinuity` and call the `retime` or `synchronize` functions, then you do not need to specify a method. Instead, `retime` and `synchronize` fill in the output `timetable` variables using different default methods for continuous, step, or event data. For more information, see the `VariableContinuity` property of `timetable` or [Retime and Synchronize Timetable Variables Using Different Methods](#).

## **mink and maxk Functions: Find the k smallest or largest elements in an array**

Finding a specified number of smallest or largest elements in array data is now possible with the `mink` and `maxk` functions.

For example, `mink(A,5)` and `maxk(A,5)` return the 5 smallest and 5 largest elements of a vector `A`, respectively.

## **topkrows Function: Find the k top rows in sorted order for numeric arrays, tables, and timetables**

The `topkrows` function adds support for in-memory numeric arrays, tables, and timetables. Previously, `topkrows` supported only tall arrays.

## **Version History**

There are a few behavior changes when `topkrows` operates on tall arrays:

- When operating on a tall array, `topkrows` places `NaN`, `NaT`, and other missing values at the end of a descending sort. In previous releases `topkrows` placed missing values at the beginning of a descending sort.
- `topkrows` no longer accepts tall cell arrays containing only scalar numeric values as inputs. Use `cell2mat` to convert the tall cell array of scalar numeric values into a tall matrix before using `topkrows`.

## App Building

### App Designer: Create apps with a wide variety of 2-D and 3-D plots

Include a wider selection of 2-D and 3-D plots in your app, such as surfaces and patches. For more information, see Graphics Support in App Designer.

### App Designer: Add menus to an app from the Component Library

Add menus to your apps by dragging a menu bar from the App Designer **Component Library** onto the canvas. Then add and edit your menu items by clicking the **+** buttons, pressing the **Delete** key, and typing directly on the menu labels.

### App Designer: Specify input arguments when running an app

Define input arguments for your app, and access them in the `StartupFcn` callback for the `UIFigure` component. You can use the input arguments to pass data between apps or to set default values in an app. See Startup Tasks and Input Arguments in App Designer for more information.

### App Designer: Add a summary, description, and screenshot for app packaging and compiling

Add a name, summary, description, and screenshot for your app. MATLAB provides this information for use in other contexts, such as:

- The **Package App** interface
- The MATLAB Compiler interface
- The file browser details on some operating systems

### App Designer: Improved component Properties pane in Code View

The component **Properties** pane in **Code View** has an improved layout with richer editing capabilities. For example, toggle buttons are now available for controlling certain properties such as font weight and text alignment.

### App Designer: Edit tick labels for gauges, knobs, and sliders directly in the canvas

Now you can change the tick labels for gauges, knobs, and sliders more quickly by editing them directly on the canvas.

### uitree and uitreenode Functions: Create trees and tree nodes in apps

Call the `uitree` and `uitreenode` functions to display lists of items in a hierarchy within your app.

To display a hierarchy in an App Designer app, call the `uitree` and `uitreenode` functions from within a callback, such as the `StartupFcn` for the `UIFigure` component. For an example, see Use App Designer to Display Items in a Tree.

The functions only work for App Designer apps or in figures created with the `uifigure` function.

### **uiconfirm Function: Create modal in-app confirmation dialog boxes**

Call the `uiconfirm` function to create a modal in-app dialog box that displays a set of options that you define. The function has an output argument that returns the user's selection.

The dialog box can only display in App Designer apps or in figures created with the `uifigure` function.

### **Toolbox Packaging: Add App Designer apps to the Apps Gallery upon toolbox installation**

When you package a toolbox, MATLAB automatically detects App Designer apps (`.mlapp` files) in addition to app installer files (`.mlappinstall` files). If you package these apps or app installer files with your toolbox, they appear in the MATLAB Apps Gallery when a user installs your toolbox. For more information, see [Create and Share Toolboxes](#).

### **MATLAB Online: Run App Designer apps in MATLAB Online**

MATLAB Online now supports running apps created in App Designer. Only the Google Chrome browser supports this capability. If you want to build an app, you must use App Designer running on a desktop version of MATLAB.

## Performance

### **App Designer: Load apps faster**

Loading apps into App Designer is 40% to 60% faster than in R2017a. The time savings becomes more noticeable as the number of components in your app increases.

### **Execution Engine: Improved performance for vectorized math on CPUs with AVX2**

The MATLAB execution engine improves performance of vectorized math on CPUs that support AVX2 instructions, compared to CPUs that support SSE2 or earlier versions of SIMD (Single-Instruction Multiple-Data) extensions.

### **Live Editor: Run live scripts with loops faster**

Live scripts containing loops run significantly faster than in previous releases. In addition, typing and scrolling in live scripts is more responsive.

## Hardware Support

### **Arduino: Wirelessly connect to Arduino boards using low-cost Bluetooth adaptors**

MATLAB Support Package for Arduino Hardware enables you to wirelessly connect to and communicate with Arduino boards over Bluetooth using low-cost Bluetooth devices such as Adafruit Bluefruit EZ-Link, HC-05, or HC-06. Instrument Control Toolbox software is required to set up Bluetooth communication.

### **Arduino Setup UI: Set up a connection to your Arduino board over USB, Bluetooth, or Wi-Fi**

MATLAB Support Package for Arduino Hardware enables you to use `arduinsetup` guided interface to set up a connection to your Arduino board over USB, Bluetooth, or Wi-Fi.

### **Arduino Plug-In Detection: Discover available Arduino support and examples when plugging a compatible Arduino board**

MATLAB Support Package for Arduino Hardware automatically detects compatible Arduino boards plugged into your computer and also shows available Arduino support for MATLAB and Simulink along with examples.

## Advanced Software Development

### **MATLAB Engine API for C++: Run MATLAB code from C++ programs with object-oriented programming support and asynchronous execution**

The MATLAB Engine API for C++ provides an interface between the C++ programming language and MATLAB. This API enables C++ programs to launch MATLAB, evaluate MATLAB functions with arguments, and exchange data between MATLAB and C++ programs. For more information, see [MATLAB Engine API for C++](#).

### **MATLAB Engine API for C++: Pass data between C++ programs and MATLAB using MATLAB Data Array**

The MATLAB Data API uses modern C++ semantics and design patterns and avoids data copies whenever possible by using MATLAB copy-on-write semantics.

### **Java SE 8: MATLAB support, providing improved security and access to new Java features**

Java interface supports JRE version Java 8. For more information, see <https://www.mathworks.com/support/requirements/language-interfaces.html>.

### **MinGW 5.3: MATLAB support**

MATLAB supports the MinGW-w64 version 5.3.0 compiler from <https://mingw-w64.org> for building MEX files and standalone MATLAB engine and MAT-file applications. To download the supported version, search for MinGW from the Add-Ons menu. If you do not install the supported version, then MATLAB displays a warning.

### **Microsoft Visual Studio 2017: MATLAB support for Microsoft Visual Studio 2017 Community, Professional, and Enterprise editions**

MATLAB supports Microsoft Visual C++ 2017 Professional, Community, and Enterprise editions for building MEX files and standalone MATLAB engine and MAT-file applications.

### **Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications**

Support	Compiler	Platform
Added	Microsoft Visual C++ 2017 Professional, Community, and Enterprise editions	Windows
Added	Microsoft Visual C++ 2015 and 2013 Community and Enterprise editions in addition to continued support for the Professional edition	Windows

Support	Compiler	Platform
Added	Microsoft Visual Studio 2015 (v140) toolset installed over Visual Studio 2017	Windows
Added	MinGW-w64 version 5.3.0 compiler from <a href="https://mingw-w64.org">https://mingw-w64.org</a>	Windows
Added	Intel Parallel Studio XE 2017 with Microsoft Visual Studio 2017 for C, C++, and Fortran	Windows
Discontinued	MinGW-w64 version 4.9.2 compiler from TDM-GCC	Windows
Discontinued	Microsoft Visual C++ 2012 Professional	Windows
Discontinued	Microsoft Windows SDK 7.1	Windows
To be phased out	Support for GNU gcc and gfortran version 4.9 will be discontinued in a future release, at which time new versions will be supported.	Linux

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see [Supported and Compatible Compilers](#).

## Python Version 3.6: MATLAB support

MATLAB supports the following versions of CPython:

- Version 2.7
- Version 3.4
- Version 3.5
- Version 3.6

For more information, see [Install Supported Python Implementation](#).

## Perl 5.24.1: MATLAB support

MATLAB ships with Perl version 5.24.1.

- See [www.perl.org](http://www.perl.org) for a standard distribution of perl, perl source, and information about using perl.
- See <https://metacpan.org/pod/HTML::Parser> for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See <https://metacpan.org/pod/HTML::Tagset> for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

## Version History

If you use the `perl` command on Windows platforms, see [www.perl.org](http://www.perl.org) for information about using this version of the Perl programming language.



## **MATLAB Handle class method: Add a listener for an event without binding the listener to the source object**

The listener handle class method adds a listener for an event without binding the listener lifecycle to the object that is the source of the event. For more information, see `handle.listener`.

## **Unit Testing Framework: Provide code coverage reports in the Cobertura format for improved continuous integration workflows**

You can specify that the code coverage plugin returns code coverage results in the Cobertura format for use within continuous integration systems. Use the 'Producing' name-value pair argument with the `forFolder`, `forPackage`, or `forFile` static method of the `matlab.unittest.plugins.CodeCoveragePlugin` class.

## **Unit Testing Framework: Generate HTML report of a test run**

Create easily readable and navigable HTML reports of your test runs using the `producingHTML` static method of the `TestReportPlugin`. For more information, see `matlab.unittest.plugins.TestReportPlugin.producingHTML`.

## **Unit Testing Framework: Write tests as live scripts**

You can write and execute script-based unit tests using Live Scripts. For more information, see `Write Test Using Live Script`.

## **Unit Testing Framework: Specify additional diagnostics to evaluate upon failures using the onFailure method**

To specify additional diagnostics for the testing framework to evaluate when a test fails, use the `onFailure` method of the `matlab.unittest.TestCase` class. For more information, see `matlab.unittest.TestCase.onFailure`.

If you create custom test fixtures, you can specify additional diagnostics for failures during fixture setup and teardown routines. For more information, see `matlab.unittest.fixtures.Fixture.onFailure`.

## **Performance Testing Framework: Define multiple measurement boundaries in test methods**

To further refine which code to measure, performance tests now support multiple measurement boundaries in a test method. Use multiple, nonnested calls to the `startMeasuring` and `stopMeasuring` methods of `matlab.perftest.TestCase` within your test methods. Measurements from multiple boundaries in the same test method are accumulated and summed. For more information, see `startMeasuring`.

## **Mocking Framework: Construct mocks for classes that have Abstract methods with other attributes**

You can construct a mock for a class that has `Abstract` methods and other attributes. For example, you can construct a mock for a class with a method that has `Abstract` and `Static` attributes. The

mock implements the method as a concrete, `Static` method. Similarly, methods with `Abstract` and `Hidden` attributes are implemented as concrete and `Hidden` methods. For more information, see [Create Mock Object](#).

## Source Control Integration: Show differences from parent files and save copies in Git Branches

In a project under Git source control, in the Branches dialog box, you can show differences to parent files and save branches. You can save a copy of the selected files or parent files to examine added or deleted files or to test how the code ran in previous versions. For more information, see [Branch and Merge with Git](#).

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Programmatic dependence on specific diagnostic subclass from <code>getDiagnosticFor</code> method of constraint and tolerances	Still runs	Not applicable	<p>Rework code that relies on properties or methods specific to <code>matlab.unittest.diagnostic.ConstraintDiagnostic</code> instances returned from the <code>getDiagnosticFor</code> method of <code>matlab.unittest.constraints</code> classes.</p> <p>In a future release, diagnostics returned from constraint and tolerance classes in the <code>matlab.unittest.constraints</code> package will be subclasses of <code>matlab.unittest.diagnostics.Diagnostic</code> and might not be instances of <code>matlab.unittest.diagnostic.ConstraintDiagnostic</code>.</p>
Negation of <code>matlab.unittest.constraints.ReturnsTrue</code> constraint ( <code>~ReturnsTrue</code> )	Warns	Not applicable	Change the logic for tests that rely on negating the <code>ReturnsTrue</code> constraint.

Functionality	Result	Use This Instead	Compatibility Considerations
Generating test suites from a class that derives from a concrete base class that defines methods that reference a <code>TestParameter</code> , <code>MethodSetupParameter</code> , or <code>ClassSetupParameter</code> not defined within the base class	Errors	Define either the: <ul style="list-style-type: none"> <li>• Base class as Abstract using the class-level Abstract attribute. For example, <code>classdef (Abstract) MyTest &lt; matlab.unittest.TestCase</code>.</li> <li>• Abstract parameter properties for all parameters used by methods of the class. For example, <code>properties (Abstract, TestParameter)</code>.</li> </ul>	If your test class inherits from a concrete base class that uses a parameter that is not defined in the base class, MATLAB throws an error.
Invoking <code>TestRunnerPlugin</code> superclass methods more than once	Errors	Not applicable	Custom plugins contain methods that invoke the corresponding <code>TestRunnerPlugin</code> superclass methods exactly once. Rework any custom plugins that invoke the superclass methods more than once. Starting in R2017b, this behavior throws an error.
Setting the event <code>ListenAccess</code> or <code>NotifyAccess</code> attributes to <code>immutable</code> .	Errors	<code>immutable</code> is not a valid value for these attributes.	In previous releases, setting these attributes to <code>immutable</code> did not cause an error.
Setting the method <code>Access</code> attribute to <code>immutable</code> .	Errors	<code>immutable</code> is not a valid value for this attribute.	In previous releases, setting this attribute to <code>immutable</code> did not cause an error.



# R2017a

---

**Version: 9.2**

**New Features**













**Bug Fixes**

**Version History**

## Desktop

### Live Editor: Edit a figure interactively including title, labels, legend, and other annotations

In the Live Editor, you can add formatting and annotations to a figure interactively. To add an item go to the **Figure** tab and in the **Annotations** section, select from these available options:

-  **Title**
-  **X-Label**,  **Y-Label**
-  **Legend**
-  **Colorbar**
-  **Grid**,  **X-Grid**,  **Y-Grid**
-  **Line**,  **Arrow**,  **Text Arrow**,  **Double Arrow**

For more information, see [Modify Figures in Live Scripts](#).

### Live Editor: Get suggestions for mistyped commands and variables

MATLAB suggests corrections for mistyped commands and variables in the Live Editor. If you enter an undefined command or variable name, MATLAB displays: `Did you mean:` followed by a suggested command under the error message. Press the **Fix** button to replace the mistyped command with the suggested fix.

### Live Editor: Copy live script outputs to other applications

When copying from the Live Editor to other applications, in line outputs within the selection are included along with the code and text. In applications that support only plain text, the code and text is copied, but outputs are not.

You also can copy an output by right-clicking the output and selecting a copy option from the context menu

### Live Editor: Hover over variables to see their current value

You can view the current value of a variable as a data tip in the Live Editor. To view the data tip, position your mouse pointer over the variable.

Data tips are enabled by default in the Live Editor. To enable or disable data tips, with a live script open in the Live Editor, go to the **View** tab, and in the **Display** section, select or deselect **Datatypes**.

## Add-On Explorer: Discover and install File Exchange submissions hosted on GitHub in Add-On Explorer

File Exchange submissions hosted on GitHub® are now available in the Add-On Explorer. For more information on how to get add-ons through the Add-On Explorer, see [Get Add-Ons](#).

## MATLAB Online: Use MATLAB through your web browser for teaching, learning, and convenient, lightweight access

Complement your MATLAB experience with MATLAB Online. For anytime, anywhere access, simply sign in with your MathWorks account — no download or installation required.

MATLAB Online is available with most MATLAB licenses. For more information including eligibility, visit the [MATLAB Online product page](#).

## Startup Folder Behavior Changes: Set initial working folder using new options and behaviors

The options to set the initial working folder (startup folder) have changed. For more information, see [MATLAB Startup Folder](#).

The behavior when starting MATLAB from a shortcut icon has changed. MATLAB starts from the last working folder, except on Linux platforms or when the **Start in** value is specified on Windows platforms.

When calling MATLAB from a command prompt, the initial working folder is the terminal window folder. To use the initial working folder set in the general preferences panel, use the `-useStartupFolderPref` startup option.

```
matlab -useStartupFolderPref
```

To set the startup folder from the command prompt, use the `-sd` option. For example:

```
matlab -sd "C:\work"
```

For information about the command line options to set the initial working folder, see `matlab` (Mac), `matlab` (Windows), and `matlab` (Linux).

## Version History

If you start MATLAB from a command window and want the startup folder to be the value set in the general preferences panel, then add the `-useStartupFolderPref` startup option.

## Language and Programming

### **string Arrays: Create string arrays using double quotes**

You can create strings using double quotes, just as you can create character vectors with single quotes.

- `str = "Hello, World"` creates a string.
- `str = ["Good" "morning"]` creates a 1-by-2 string array.

For more information, see [Characters and Strings](#).

### **String Functions: Return character arrays or cell arrays instead of string arrays**

The data type of the output argument has changed for the `compose`, `extractAfter`, `extractBefore`, `extractBetween`, `join`, `split`, and `splitlines` functions. As of R2017a, the output data type depends on the type of the first input argument.

- If the first input argument is a string array, then the output argument is a string array.
- If the first input argument is a character vector, then the output argument is either a character vector or a cell array of character vectors.
- If the first input argument is a cell array of character vectors, then the output argument is either a character vector or a cell array of character vectors.

In R2016b, these functions always return string arrays.

### **missing Function: Assign missing values in core data types, including double, datetime, categorical, and string arrays**

To assign missing values to elements of an array or table, use the `missing` function. For example, `catarray(3) = missing` assigns the third element of a categorical array `catarray` to a missing value.

### **issortedrows Function: Determine if matrix and table rows are sorted**

Use the `issortedrows` function to determine if matrix or table rows are sorted. For example, `issortedrows(A, 'descend')` checks if the rows of a matrix `A` are in descending order based on the elements in the first column.

### **sort and sortrows Functions: Specify options for sorting complex numbers and placing missing elements**

Indicate how you want to sort complex numbers and place missing elements using the `sort` and `sortrows` functions with the `'ComparisonMethod'` and `'MissingPlacement'` options. For example, if `A` is a numeric vector containing complex numbers, then `sort(A, 'ComparisonMethod', 'real', 'MissingPlacement', 'last')` sorts by the real part of each element and places NaN values at the end of the sort.



## issorted Function: Query sort order with monotonic, strictly monotonic, strictly ascending, and strictly descending options

You can determine if an array is sorted using the 'monotonic', 'strictmonotonic', 'strictascend', and 'strictdescend' options with the `issorted` function. For example, `issorted(A, 'strictmonotonic')` checks if a vector `A` is strictly monotonic.

## head and tail Functions: Return top or bottom rows of table or timetable

To return the top or bottom rows of a table or a timetable, you can use the `head` and `tail` functions.

## table Data Containers: Use row labels when performing join, sort, and grouping operations

When you join tables, sort tables, or use grouping variables, you can specify row labels and table variables together as key variables, sorting variables, or grouping variables. You can specify them in the `innerjoin`, `join`, `outerjoin`, `rowfun`, `sortrows`, `stack`, `unstack`, and `varfun` functions. In previous releases, you could specify row labels or table variables, but not both together.

Row labels are the row names of a table or the row times of a timetable.

The `innerjoin`, `join`, and `outerjoin` functions have limitations on key variables:

- You cannot perform an inner or outer join using row labels as the left key and a table variable as the right key. To perform the inner or outer join, convert the row labels to a table variable and use the new variable as a key.
- You cannot perform an outer join using both row labels and table variables as key variables. To perform the outer join, convert the row labels to a table variable, or use the row labels as the only key variable.
- You cannot specify a table as the first input when you perform an inner or outer join on a table and a timetable. The timetable must be the first input.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
compose extractAfter extractBefore extractBetween join split splitlines	Still runs	Not applicable	Starting in R2017a, the output data type depends on the data type of the first input argument. In R2016b, these functions always return string arrays. For more information, see “String Functions: Return character arrays or cell arrays instead of string arrays” on page 11-4.

Functionality	Result	Use This Instead	Compatibility Considerations
issorted using 'rows' option with matrix input	Still runs	issortedrows	To determine whether the rows of a matrix are sorted, use the <code>issortedrows</code> function. <code>issortedrows</code> provides more capability for row sort queries, such as whether the rows are in ascending or descending order.
issorted with timetable input	Still runs	issortedrows	To determine whether row times or table variables of a timetable are sorted, use the <code>issortedrows</code> function. <code>issortedrows</code> provides more capability for timetable sort queries, such as whether row times are in ascending or descending order.
innerjoin join outerjoin rowfun sortrows stack unstack varfun	Still runs	Not applicable	Starting in R2017a, you can specify row labels and table variables together as key variables, sorting variables, or grouping variables. In previous releases, you could specify row labels or table variables, but not both together. For more information, see “table Data Containers: Use row labels when performing join, sort, and grouping operations” on page 11-5.
notebook	Warns	Live Editor	In future releases, calling <code>notebook</code> returns an error.

Functionality	Result	Use This Instead	Compatibility Considerations
dir	Still runs	Not applicable	<p>Starting in R2017a on UNIX platforms, if your current working folder is invalid and you call the <code>dir</code> function with a file or folder name, MATLAB throws an error. Your working folder becomes invalid if you delete it while it is your current working folder.</p> <p>In previous versions of MATLAB, if your current working folder was invalid and you called <code>dir</code> with an input, <code>dir</code> returned empty.</p>

## Graphics

### heatmap Function: Visualize table or matrix data as a heatmap

To create a heatmap chart, use the heatmap function.

### legend Function: Create legends that update when data is added to or removed from the axes

Legends now automatically update when you add or remove graphics objects from the axes. Previously, legends did not automatically add items for new graphics objects or remove items for deleted graphics objects.

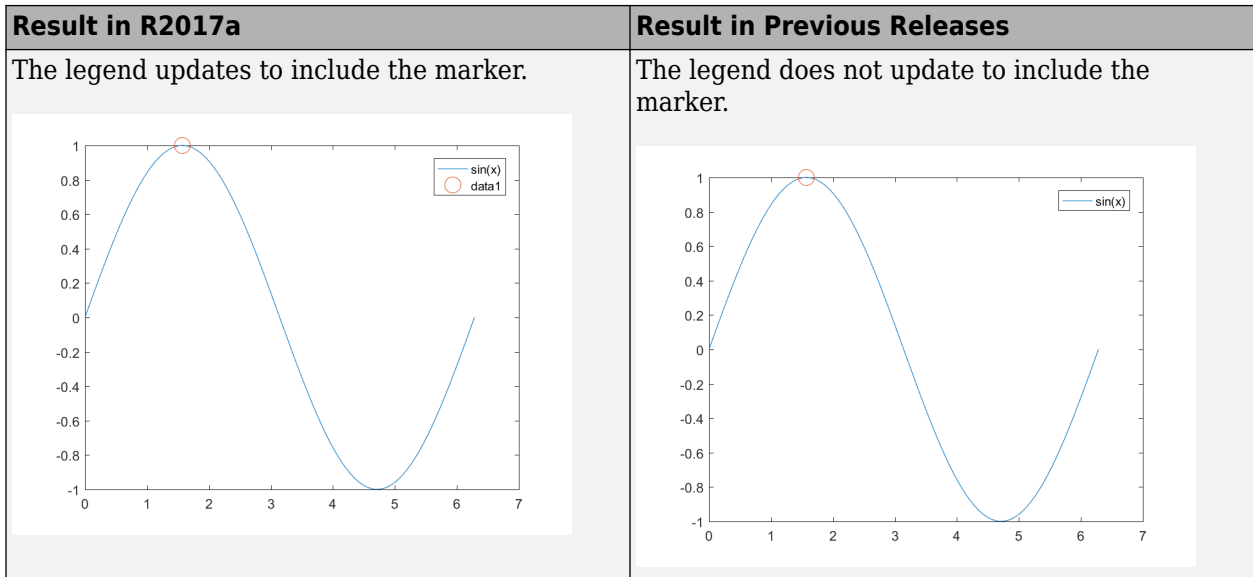
For example, this code plots a line and adds a legend. Then the code plots a second line. The legend automatically updates to include the second line.

```
plot(1:10)
legend('Line 1')
hold on
plot(11:20, 'DisplayName', 'Line 2')
hold off
```

### Version History

If you add or delete a graphics object after creating a legend, the number of items in the legend can differ from previous releases. For example, plot a line and create a legend. The plot displays a marker at the maximum point.

```
x = linspace(0,2*pi);
y = sin(x);
plot(x,y)
legend('sin(x)')
hold on
plot(pi/2, sin(pi/2), 'o', 'MarkerSize', 12)
hold off
```



If you do not want the legend to update automatically, set the `AutoUpdate` property of the legend to `'off'`. Graphics objects added to the axes after the legend is created do not appear in the legend.

```
legend('sin(x)', 'AutoUpdate', 'off')
```

To change the behavior for newly created legends, set the default value of the `AutoUpdate` property. To affect all new legends, set the value on the root level. Alternatively, to affect all new legends in a particular figure, set the value on the figure level. After you set the default value, any new legends have an `AutoUpdate` property set to `'off'`.

```
fig = figure;
set(fig, 'defaultLegendAutoUpdate', 'off')
```

Additional options for excluding specific graphics objects from a legend include:

- Specifying the graphics objects to include in the legend as an input argument to the `legend` function. For example, this code creates a legend that includes only the graphics objects referred to by `p1` and `p2`. However, graphics objects added to the axes after the legend is created do not appear in the legend. Consider creating the legend after creating all the plots to avoid extra items.

```
p1 = plot(1:10);
hold on
p2 = plot(11:20);
p3 = plot(21:30);
legend([p1 p2], 'line 1', 'line 2')
```

- Excluding graphics objects from the legend by setting their `HandleVisibility` property to `'off'`. The `HandleVisibility` property also controls the visibility of the graphics object in the `Children` property of its parent.

```
p4 = plot(31:40, 'HandleVisibility', 'off');
```

- Excluding graphics objects from the legend by setting the `IconDisplayStyle` property of the underlying annotation object to `'off'`.

```
p5 = plot(41:50);
p5.Annotation.LegendInformation.IconDisplayStyle = 'off';
```

## Categorical Plotting: Use categorical data in common plotting functions and customize axes with categorical rulers

These graphics functions accept input data of type `categorical`.

<code>bar</code>	<code>barh</code>
<code>plot</code>	<code>plot3</code>
<code>semilogx</code> (x values must be numeric)	<code>semilogy</code> (y values must be numeric)
<code>stem</code>	<code>stairs</code>
<code>scatter</code>	<code>scatter3</code>
<code>area</code>	<code>mesh</code>
<code>surf</code>	<code>surface</code>
<code>fill</code>	<code>fill3</code>
<code>line</code>	<code>text</code>

Also, you can use the new categorical ruler to customize the axes. Access the `CategoricalRuler` object through the `XAxis`, `YAxis`, or `ZAxis` property of the axes object. For example:

```
c = categorical({'red','yellow','blue'});
y = [1 2 3];
plot(c,y)
ax = gca;
ax.XAxis
```

ans =

```
CategoricalRuler with properties:
```

```
Categories: [blue    red    yellow]
Limits:     [blue    yellow]
TickValues: [blue    red    yellow]
```

```
Show all properties
```

For a complete list of properties, see `CategoricalRuler`.

### histogram Function: Plot histograms of datetime and duration data

`histogram` accepts input data of type `datetime` and `duration`. Additionally, you can bin the data using units of time as the bin edges, such as `'second'`, `'hour'`, or `'week'`.

### histogram Function: Sort categorical bins by bar height, and limit the number of bins displayed

`histogram` can sort categorical bins by bar height (highest and lowest), and additionally it can limit the number of bars displayed. This functionality is most useful with data sets that have a large number of categories, since it enables you to, for example, plot only the 10 largest bars or only the 15 smallest bars.

## scatter Function: Create scatter plots of datetime and duration data

scatter and scatter3 accept input data of type datetime and duration.



## Scatter Plots: Create scatter plots with varying marker sizes faster

When you create a scatter plot and specify the marker sizes using the SizeData property, the scatter plot renders with improved performance.

## parula Colormap: Create plots with enhanced colors

The default parula colormap has enhanced colors that are more perceptually uniform.

This table shows a comparison between the R2017a and R2016b versions of the parula colormap. The visual change is subtle; however, you might notice more colorful colors and smoother transitions between colors.

R2017a	
R2016b	

## Version History

Plots that use the parula colormap maintain their overall visual appearance. However, the array of colors returned by the parula function is different. If your code relies on the specific RGB triplet values returned by parula, then you might need to update your code.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
legend function	Still runs	Not applicable	Legends now automatically update to include new graphics objects added to the axes. Similarly, they update to exclude graphics objects deleted from the axes. For more information, see “legend Function: Create legends that update when data is added to or removed from the axes” on page 11-8.

Functionality	Result	Use This Instead	Compatibility Considerations
parula function	Still runs	Not applicable	Plots that use the parula colormap maintain their overall visual appearance. However, the array returned by the parula function contains different RGB triplet values. For more information, see “parula Colormap: Create plots with enhanced colors” on page 11-11.
Normalization option of histogram and histogram2	Still runs	Not applicable	<p>The Normalization option of histogram and histogram2 now computes the normalization using the total number of input data elements. Any data that is not binned (for example NaN values) or that falls outside the bin limits is still counted for the purposes of normalization.</p> <p>Previously, the normalization used only binned data in the calculation, which can be less than the total number of input data elements if the data contains NaNs or some data is outside the bin limits. For an example of how to achieve the old behavior, see Control Categorical Histogram Display.</p>



Functionality	Result	Use This Instead	Compatibility Considerations
area function	Still runs	Not applicable	<p>The <code>area</code> function now sorts the inputted data in order of increasing <math>x</math> values before plotting. Previously, the <code>area</code> function plotted the data in the order specified, which sometimes resulted in overlapping areas. The values of the <code>XData</code> and <code>YData</code> properties of the <code>Area</code> object are unchanged.</p> <p>For example, <code>area([3 1 2],[0 5 10])</code> now draws a line from <math>(1,5)</math> to <math>(2,10)</math> to <math>(3,0)</math> and fills in the area below the line. Previously, it drew a line from <math>(3,0)</math> to <math>(1,5)</math> to <math>(2,10)</math>, resulting in overlapping areas. If you want to draw filled polygons, consider using the <code>fill</code> function instead.</p>

## Data Import and Export

### **datastore and tabularTextDatastore Functions: Automatically detect and return date and time data in text files**

The `datastore` and `tabularTextDatastore` functions detect and return date and time data as `datetime` type.

#### **Version History**

Previously, `datastore` and `tabularTextDatastore` functions returned date and time data as character vectors. To preserve that behavior in the `datastore` function, use:

```
ds = datastore(location, 'DatetimeType', 'text')
```

To preserve the old behavior in the `tabularTextDatastore` function, use:

```
ds = tabularTextDatastore(location, 'DatetimeType', 'text')
```

### **datastore Function: Work with data in Amazon S3 cloud storage**

Create an `ImageDatastore`, `FileDatastore`, or `TabularTextDatastore` to work with data stored in the cloud using Amazon S3 (Simple Storage Service). For details, see [Read Remote Data](#).

### **Import Tool: Import strings and categorical arrays interactively**

The import tool now supports importing text as `string` and `categorical` data types. For more information, see [Import Tool](#).

#### **Version History**

Previously, the Import Tool imported text data as a cell array of character vectors. To preserve that behavior, change settings in the **Text Options** field in the **Imported Data** section of the **Import** tab. Click the **Text Options** field and change the text type selection from **String Array** to **Cell Array of Character Vectors**.

### **detectImportOptions Function: Control import properties of fixed-width text files**

Control and customize how data is imported from fixed-width text files using the `detectImportOptions` function and creating a `FixedWidthImportOptions` object.

In addition to the `SpreadsheetImportOptions` and `DelimitedTextImportOptions` objects, the `detectImportOptions` function now returns a `FixedWidthImportOptions` object. Use the `FixedWidthImportOptions` object with `readtable` to customize import options, such as:

- Import bad or missing data.
- Import only a subset of data using the `SelectedVariableNames` property.
- Manage the import of partial fields.

For more information, see `FixedWidthImportOptions`.

## **RESTful web services: Support for PUT and DELETE HTTP methods in webread, webwrite, and websave**

To call HTTP PUT, DELETE, and PATCH methods, use the 'RequestMethod' argument in the `weboptions` function.

## **save Function: Save workspace variables to a MAT-file with or without compression**

Previously, the `save` command, when saving workspace variables to a MAT-file, used compression as the default (and the only) option. Now, a new option, '-nocompression', allows saving of data without compression. This option is only available to use with MAT-File version 7.3.

By default, saving a variable `myVariable` to a MAT-file in version 7.3 compresses the data:

```
save -v7.3 myFile.mat myVariable
```

To save `myVariable` without compression, use:

```
save -v7.3 -nocompression myFile.mat myVariable
```

## **writetable Function: Select preferred character encoding when writing to a file**

Now, you can specify a character encoding of your choice by using the 'Encoding' parameter. Previously, the `writetable` function used the system's default encoding when writing to a file and that was the only available option. For example, to set file encoding to support Japanese characters, set the `Encoding` parameter to `Shift_JIS`.

```
writetable(T, 'filename.csv', 'Encoding', 'Shift_JIS');
```

## **NetCDF Functions: Create variable names and attributes containing non-ASCII characters**

NetCDF functions now support the use of non-ASCII characters for variable names, attribute names, and attribute values. For more information on creating NetCDF files see, `NetCDF Files and netcdf`

## **Webcam Support Package: GStreamer Upgrade on Linux**

The MATLAB Support Package for USB Webcams Linux support now uses the GStreamer library version 1.0. It previously used version .10.

## **jsondecode converts JSON null values in numeric arrays to NaN**

The `jsondecode` function converts JSON null values in numeric arrays to NaN. For nonnumeric arrays, this function converts a JSON null value to an empty double array (`[]`). This behavior affects the `webread` and `webwrite` functions when processing JSON content.

## Version History

Previously, `jsondecode` decoded a JSON `null` value as an empty double array, which is dropped from the output. For example:

```
json = '[1, 2, null, 4]';
res = jsondecode(json) % R2016b
```

```
res =
     1
     2
     4
```

The current behavior is:

```
json = '[1, 2, null, 4]';
res = jsondecode(json) % R2017a and later
```

```
res =
     1
     2
    NaN
     4
```

To preserve the old behavior in the `jsondecode` function, that is, to remove the NaN values from the resulting numeric array, use the `rmmissing` function:

```
res = rmmissing(res)
```

```
res =
     1
     2
     4
```

## load and fopen Functions: Use the file separator character ('\') preceding a file name to indicate that the file is in the root folder

Both the `load` and `fopen` functions take `filename` as an input argument. When specifying this `filename`, use the file separator character preceding a file name to indicate that the file is in the root folder. The file separator character is `'\'` on Windows and `'/'` on UNIX. For example, if you specify `filename` as `'\myfile.txt'`, then MATLAB searches the root folder for this file.

- If `myfile.txt` exists in the root folder, then MATLAB proceeds with loading or opening the file.
- If `myfile.txt` does not exist in the root folder, then the `load` function returns an error, and the `fopen` function returns a -1 indicating that the file could not be opened.
- If `myfile.txt` does not exist in the root folder, but a file with the same name exists in the current folder, then MATLAB behaves in the same way as the previous step. That is, the `load` function returns an error, and the `fopen` function returns a -1 indicating that the file could not be opened, because no such file exists in the root folder.

## Version History

Previously, for the `load` and `fopen` functions, if you specified a filename as `'\myFile.txt'`, then MATLAB would look for that file in your root folder. If no such file was found in the root folder, then MATLAB would attempt to find and return the file from the current folder. However, now, if you add a file separator preceding the file name, then the `load` and `fopen` functions will only look for the file in the root folder. Therefore,

- To indicate that file is located in root folder use `load('\myFile.mat')` or `fopen('\myFile.mat')`.
- To indicate that file is located in current folder use `load('myFile.mat')` or `fopen('myFile.mat')`.

Additionally, on UNIX platforms, if you use `'/'` as the root symbol when specifying the file name, and the file is on the user path, then the path portion specified must case match. That is, MATLAB no longer case-corrects specified paths that begin with the root symbol `'/'`. For example, if a file `'/user/myFile.mat'` exists under the root folder. Then, on UNIX platforms:

- `load('/user/myFile.mat')` or `fopen('/user/myFile.mat')` works correctly.
- `load('/User/myFile.mat')` or `fopen('/User/myFile.mat')` does not work due to the case mismatch in the specified path.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Reading video files, with old file formats, on macOS platforms using <code>VideoReader</code>	Errors	Not applicable	<p>The macOS platform no longer supports certain older video file formats. To read such files using <code>VideoReader</code>:</p> <ul style="list-style-type: none"> <li>• Open the video file using the QuickTime player. If the file is supported on macOS, the player automatically converts the file into a newer format.</li> <li>• Save this newly converted video file.</li> <li>• Use <code>VideoReader</code> to read this new converted video file.</li> </ul>

## Data Analysis

### tall Arrays: Operate on tall arrays with more functions, including ismember, sort, conv, and moving statistics functions

Various functions add support for tall arrays as inputs.

array2table	cumprod	movmad	movstd	sort
bsxfun	cumsum	movmax	movsum	sortrows
cell2mat	diff	movmean	movvar	table2timetable
conv	ismember	movmedian	pie (categorical only)	timetable
cummax	issorted	movmin	repelem	timetable2table
cummin	issortedrows	movprod	repmat	varfun

For information about limitations of this support, type `help tall/<function>` in MATLAB, or see [Functions That Support Tall Arrays \(A-Z\)](#).

### tall Arrays: Index tall arrays using sorted indices

Now to index tall arrays, you can use ascending or descending sorted indices. The indices can specify elements anywhere in the array, and allow for duplicates. For example:

- `T(0:2:100)`
- `T([1 2 2 5 20])`
- `T(100:-10:50)`

For more information, see [Index and View Tall Array Elements](#).

### tall Arrays: Work with out-of-memory, time-stamped data in a timetable

Convert a tall table containing a time variable into a tall `timetable` to facilitate calculations on large sets of time-stamped data. For more information, see [Functions That Support Tall Arrays \(A-Z\)](#).

### isoutlier and filloutliers Functions: Detect and replace outliers in an array or table

To find outliers in your data, use the `isoutlier` function. To replace outliers with alternative values, use the `filloutliers` function.

## **smoothdata Function: Smooth noisy data in an array or table with filtering or local regression**

Smoothing noisy data is now possible with the `smoothdata` function. For example, `smoothdata(A, 'movmedian')` smooths data with a moving-window median.

## **summary Function: Calculate summary statistics and variable information in tables and timetables**

To return a structure that contains a summary of a table or a timetable, use the `summary` function.

## **histcounts Function: Bin datetime and duration data**

`histcounts` accepts input data of type `datetime` and `duration`. Also, you can bin the data using units of time as the bin edges, such as `'second'`, `'hour'`, or `'week'`.

## **movmad and movprod Functions: Compute moving median absolute deviation and moving product of an array**

Use a sliding window to compute the moving median absolute deviation and the moving product along data in an array with the `movmad` and `movprod` functions.

## **bounds Function: Simultaneously determine the smallest and largest elements of an array**

Find the smallest and largest elements of an array with the `bounds` function.

## **fillmissing Function: Replace missing data in an array or table using moving mean or moving median option**

Filling missing data using a moving mean or moving median option is now available with the `fillmissing` function.

## **Moving Statistics Functions: Supply sample points for time-stamped and nonuniform data in moving statistics functions, such as movmean**

Providing sample points that represent the location of data in an array is now possible when computing moving statistics with the functions `movmad`, `movmax`, `movmean`, `movmedian`, `movmin`, `movprod`, `movstd`, `movsum`, and `movvar`. For example, you can compute the moving-window average of data in an array `A` with respect to times in a vector `t` using `movmean(A, 'SamplePoints', t)`.

## **prod and cumprod Functions: Ignore NaNs using 'omitnan'**

Now you can exclude NaNs when calculating the product and cumulative product of an array with the `prod` and `cumprod` functions.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Normalization option of <code>histcounts</code> and <code>histcounts2</code>	Still runs	Not applicable	<p>The <code>Normalization</code> option of <code>histcounts</code> and <code>histcounts2</code> now computes the normalization using the total number of input data elements. Any data that is not binned (for example <code>NaN</code> values) or that falls outside the bin limits is still counted for the purposes of normalization.</p> <p>Previously, the normalization used only binned data in the calculation, which can be less than the total number of input data elements if the data contains <code>NaN</code>s or some data is outside the bin limits. For an example of how to achieve the old behavior, see <code>Control Categorical Histogram Display</code>.</p>
Support for running MATLAB MapReduce on Hadoop 1.x clusters will be removed in a future release.	Warns	Use clusters that have Hadoop 2.x or higher installed to run MATLAB MapReduce.	Migrate MATLAB MapReduce code that runs on Hadoop 1.x to Hadoop 2.x.



## App Building

### App Designer: Learn to build apps using an interactive tutorial

Learn how to build a simple app using an interactive tutorial that guides you through each step in the process. You can access this tutorial in the App Designer **Open** menu.

### App Designer: Zoom and pan plots

Enable zooming and panning for plots in your apps using the `zoom` and `pan` functions. To enable this functionality, add buttons to your app that call `zoom` and `pan` in their callbacks. For more information, see [Graphics Support in App Designer](#).

### App Designer: Configure columns of a table to automatically fill the entire width of the table

MATLAB automatically calculates column widths so that they expand to fill all available space within the width of the table. This behavior is enabled by default, or whenever the `ColumnWidth` property of the `Table` UI component is set to `'auto'` for one or more columns.

### App Designer: Manage common design-time settings using the Preferences dialog box

Specify which options are always enabled or disabled whenever you work in App Designer. For more information, see [App Designer Preferences](#).

### App Designer: Include comet, graph, and digraph visualizations in apps

Use `comet` to display comet plots, and use the `graph plot` function to display graph and digraph plots. For more information, see [Graphics Support in App Designer](#).

### App Designer: Write `ButtonDownFcn` callbacks for graphics objects displayed in UI axes

Create apps containing interactive plots by writing `ButtonDownFcn` callbacks for graphics objects such as `Line` or `Bar` objects. Write the callback in a separate program file that is on the MATLAB path. For example, here is a `ButtonDownFcn` callback saved as `mybuttondown.m`.

```
function mybuttondown(src,evt)
    src.Color = rand(1,3);
end
```

To create a plot line that changes color when the user clicks the line, add this code to any callback in App Designer.

```
plot(app.UIAxes,1:10,'ButtonDownFcn',@mybuttondown);
```

**Note** The `ButtonDownFcn` supports 2-D graphics objects that are children of the `UIAxes` (such as `Line`, `Bar`, and `Scatter`). However, `UIAxes` objects and `Figure` objects created with the `uifigure` function do not support the `ButtonDownFcn`.

---

## App Designer: Edit table column headings directly in the canvas

Now you can edit table column headings directly in the canvas. The **Uitable Properties** panel automatically reflects your changes.

## App Designer: Disable automatic resize behavior when writing `SizeChangedFcn` callbacks

`SizeChangedFcn` callbacks no longer execute when the automatic resize behavior is enabled. The automatic resize behavior is enabled when the **Resize components when app is resized** check box in the **UI Figure Properties** panel (Design View) is checked.

## Version History

In previous releases, the presence of a `SizeChangedFcn` callback disabled the automatic resize behavior. The `SizeChangedFcn` callback executed whenever the container's size changed (regardless of the state of the **Resize components when app is resized** check box). Now, the automatic resize behavior must be disabled to allow the `SizeChangedFcn` to execute. Apps created in previous releases will not execute the `SizeChangedFcn` callback if the automatic resize behavior is enabled.

To disable automatic resize behavior:

- 1 Open your app in App Designer, and select **Design View**.
- 2 Click a blank area of the canvas to select the UI figure.
- 3 In the **UI Figure Properties** panel, deselect the **Resize components when app is resized** check box.
- 4 Save and run your app.

## Performance

### **Execution Engine: Improved performance for setting MATLAB object properties**

MATLAB has improved the performance of property set methods. For more information on these methods, see Property Set Methods.

### **save Function: Save MAT v7.3 files without compression for improved performance on some storage devices**

You can improve performance on some storage devices by saving variables to MAT-File version 7.3 without compression. To save a variable (`myVariable`) without compression to a MAT-file (`myFile.mat`), use:

```
save -v7.3 -nocompression myFile.mat myVariable
```

For more information, see `save`.

### **memoize Function: Cache results of a function to avoid rerunning when called with the same inputs**

*Memoization* is an optimization technique used to speed up programs by storing the results of expensive function calls and returning the cached result when the program is called with the same inputs. For more information, see `memoize`.

### **Scripts: Improved performance of scripts with lower script overhead**

MATLAB has improved the performance of invoking scripts, especially when invoking a script from another script.

### **try, catch Block: Improved performance of try blocks with lower execution overhead**

`try`, `catch` blocks that do not generate errors or throw exceptions now have lower execution overhead.

### **App Designer: Load apps faster**

Loading apps into App Designer is 20% to 35% faster. The time savings becomes more noticeable as the number of components in your app increases.

### **Mathematics Functions: Various performance improvements**

The following mathematics functionality shows improved performance:

- The backslash command `A\B` is faster when operating on negative definite matrices.

- The transposed backslash command `A' \ B` for sparse `A` is generally faster, especially for triangular matrices.
- `unique` executes faster when the input data is sorted.
- `kron` shows improved performance with sparse matrices.
- Dimensional reduction functions (`sum`, `prod`, `any`, `all`, and so on) show improved performance when operating on the second dimension (`dim = 2`) of sparse matrices.

## Hardware Support

### **Arduino: Read from quadrature encoders**

The MATLAB Support Package for Arduino Hardware enables you to read from quadrature encoders to determine the speed, acceleration, and position of a rotating device.

### **Arduino: Wirelessly connect to Arduino MKR1000 board over Wi-Fi**

The MATLAB Support Package for Arduino Hardware enables you to wirelessly connect and communicate to the Arduino MKR1000 board over Wi-Fi.

## Advanced Software Development

### Class `matlab.lang.OnOffSwitchState`: Represent on and off as logical values

The `matlab.lang.OnOffSwitchState` enumeration class provides better logical compatibility for properties and function arguments that have a natural “on” or “off” representation. This class enables properties that represent on/off states to accept logical values (`true`, `false`, `1`, `0`), as well as the character vectors `'on'` and `'off'` and the enumerations `on` and `off`.

### Object Properties: Validate object property values by their type, size, shape, or other parameters

Class definitions for properties enable you to specify size, class, and other criteria that MATLAB uses to validate values assigned to properties. For more information, see [Validate Property Values](#).

### Validation Functions: Validate that values meet specific criteria by calling the appropriate function

Validation functions determine if values meet specific criteria and return descriptive error messages if the values do not satisfy these criteria. The primary use for validation functions is for property validation. For more information on using validation functions, see [Property Validation Functions](#).

Name	Meaning
<code>mustBePositive(A)</code>	$A > 0$
<code>mustBeNonpositive(A)</code>	$A \leq 0$
<code>mustBeFinite(A)</code>	A has no NaN and no Inf elements.
<code>mustBeNonNan(A)</code>	A has no NaN elements.
<code>mustBeNonnegative(A)</code>	$A \geq 0$
<code>mustBeNegative(A)</code>	$A < 0$
<code>mustBeNonzero(A)</code>	$A \neq 0$
<code>mustBeGreaterThan(A,B)</code>	$A > B$
<code>mustBeLessThan(A,B)</code>	$A < B$
<code>mustBeGreaterThanOrEqual(A,B)</code>	$A \geq B$
<code>mustBeLessThanOrEqual(A,B)</code>	$A \leq B$
<code>mustBeNonempty(A)</code>	A is not empty
<code>mustBeNonsparse(A)</code>	A has no sparse elements.
<code>mustBeNumeric(A)</code>	A is numeric.
<code>mustBeNumericOrLogical(A)</code>	A is numeric or logical.
<code>mustBeReal(A)</code>	A has no imaginary part.
<code>mustBeInteger(A)</code>	$A == \text{floor}(A)$
<code>mustBeMember(A,B)</code>	A is an exact match for a member of B.

## **Mocking Framework: Isolate a portion of a system to test by imitating behavior of dependent components**

When unit testing, you are often interested in testing a portion of a complete system isolated from dependent components. To imitate behavior of dependent components, use the mocking framework. For more information, see [Mocking Framework](#).

## **Unit Testing Framework: Generate screenshots and figures during testing with `ScreenshotDiagnostic` and `FigureDiagnostic`**

To record screenshots and save figures, pass `ScreenshotDiagnostic` and `FigureDiagnostic` instances to test qualifications or the `log` method of the `TestCase` class. For example, within a test, `testCase.verifyEqual(actual, expected, ScreenshotDiagnostic)` captures a screen shot when `actual` is not equal to `expected`. By default, MATLAB only records diagnostics on qualification failures. However, you can record passing diagnostics by configuring the `TestRunner` with a plugin such as the `TestReportPlugin` or `DiagnosticsRecordingPlugin`. To produce the artifacts independently from qualification failures, use the `TestCase.log` method.

You can specify where MATLAB stores these artifacts using the `ArtifactsRootFolder` property of a test runner. For more information, see `matlab.unittest.TestRunner`.

## **Version History**

The `DiagnosticResult` property name has changed to `DiagnosticText` in the following classes in the `matlab.unittest.diagnostics` package: `Diagnostic`, `ConstraintDiagnostic`, `DisplayDiagnostic`, and `FunctionHandleDiagnostic`.

The `TestDiagnosticResult` and `FrameworkDiagnosticResult` properties in the `matlab.qualifications.QualificationEventData` class have been removed. They are replaced respectively by `TestDiagnosticResults` and `FrameworkDiagnosticResults` properties that contain `DiagnosticResult` objects. The properties they replace were cell arrays of character vectors.

The `DiagnosticResult` property in the `matlab.diagnostics.LoggedDiagnosticEventData` class has been removed. It is replaced by the `DiagnosticResults` property that contains `DiagnosticResult` objects. The `DiagnosticResult` property was a cell arrays of character vectors.

## **Unit Testing Framework: Capture screenshots and figures generated during tests using `TestReportPlugin`**

If your tests generate screenshots and figures during tests, they are included in a test report generated with the `TestReportPlugin` class. By default, MATLAB stores only the artifacts associated with failed diagnostics. However, you can indicate that your test report includes passing diagnostics, or explicitly log the artifacts in your test using the `TestCase.log` method. For more information, see `matlab.unittest.plugins.TestReportPlugin`.

## Unit Testing Framework: Control `runtests` function with `debug`, `strict`, and `verbosity` options

The `runtests` function has additional options:

- To pause test execution and enter debug mode in the event of a test failure, use the 'Debug' option.
- To generate a qualification failure if MATLAB issues a warning during the execution of a test, use the 'Strict' option.
- To run tests at different verbosity levels, use the 'Verbosity' option.

For more information, see `runtests`.

## Unit Testing Framework: Select tests by procedure name

You can run tests with a specified procedure name. The procedure name is different from the test element name because it does not include any class or package name or information about parameterization. In a class-based test, the procedure name is the name of the test method. In a function-based test, it is the name of the local function that contains the test. In a script-based test, it is a name generated from the test section title. For example, if a test element name for a test with parameterization is `MyTestClass/myTestMethod(param1=val1,param2=valB)` the procedure name is `myTestMethod`.

- To select and run test elements with a specified procedure name, use the 'ProcedureName' name-value pair with the `runtests`, `runperf`, and `testsuite` functions or the `TestSuite` suite creation methods.
- To select test elements from an existing test suite, use the `TestSuite.selectIf` method with the `HasProcedureName` selector.

## Unit Testing Framework: Comparator for MATLAB tables

You can use the `TableComparator` class with the `IsEqualTo` constraint to compare table values recursively.

## Version History

The `verifyEqual`, `assertEqual`, `assumeEqual`, and `fatalAssertEqual` methods and `IsEqualTo` constraint compare MATLAB tables using the `TableComparator` class. Prior to R2017a, they used the `ObjectComparator`, which is less strict. Therefore, it is possible that if you have tests that compare classes, test that used to pass might now fail.

## Performance Testing Framework: View statistics from test measurements with the `sampleSummary` method

To create a table of summary statistics from the results for running a measurement experiment on a test suite, use the `sampleSummary` method of the `MeasurementResult` class. For more information, see `matlab.unittest.measurement.MeasurementResult.sampleSummary`.



## Performance Testing Framework: Apply a function across test measurements with the samplefun method

To apply a function across the Samples of a MeasurementResult array, use the samplefun method. For more information, see `matlab.unittest.measurement.MeasurementResult.samplefun`.

## Source Control Integration: Use Git Pull to fetch and merge in one step

You can now use Pull for Git source control integration. Pull fetches the latest changes and merges them into your current branch. Previously, you had to fetch and merge separately before you could see changes. For more information, see Pull, Push and Fetch Files with Git.

## MEX builds with 64-Bit API by default

The mex function uses the large-array-handling API (`-largeArrayDims` option) by default. Best practice is to update your MEX source code to use this library and rebuild the MEX file. For instructions, see Upgrade MEX Files to Use 64-Bit API.

You can run existing binary MEX files without rebuilding. For more information, see Version Compatibility.

If you build MEX files without using the mex command options `-largeArrayDims` or `-compatibleArrayDims`, then review the table in Compatibility Considerations to avoid depending on default behavior that changes in R2017a. For information about the consequences of using the `-compatibleArrayDims` option to build MEX files, see What If I Do Not Upgrade?

The default build mode for C MEX S-functions remains `-compatibleArrayDims`.

## Version History

This table shows the changes you must make to your mex command to build existing MEX files or S-functions.

Source Code	mex command — R2016b and earlier	mex command — R2017a and later
MEX file C/C++ or Fortran source code uses 32-bit API	<code>mex myMex.c</code>	<code>mex myMex.c -compatibleArrayDims</code>
	<code>mex myMex.c -compatibleArrayDims</code>	No change.
MEX file C/C++ or Fortran source code uses 64-bit API	<code>mex myMex.c -largeArrayDims</code>	Use: <code>mex myMex.c</code>  Or: <code>mex myMex.c -largeArrayDims</code>

Source Code	mex command — R2016b and earlier	mex command — R2017a and later
S-function C/C++ source code uses 32-bit API	mex sfun.c	No change.
	mex sfun.c -compatibleArrayDims	No change.
S-function C/C++ source code uses 64-bit API	mex sfun.c -largeArrayDims	No change.
S-function Fortran source code uses 32-bit API	mex sfun.F	mex sfun.F -compatibleArrayDims
S-function Fortran source code uses 64-bit API	mex sfun.F -largeArrayDims	No change.

## MEX files and shared libraries: Diagnostic information displayed for failure to load

Instead of displaying `Module Not Found` or `Invalid MEX-file` errors, MATLAB displays diagnostic information. For more information, see [Invalid MEX File Errors and Loading Library Errors](#).

## Java: Supports string data type

When calling a Java function, MATLAB converts `string` scalar arguments to `java.lang.String` and `string` array arguments to `java.lang.String[]`. For more information, see [Pass Data to Java Methods](#).

The MATLAB `string` function converts `java.lang.String` scalar arguments to a `string` scalar and converts a `java.lang.String[...]` argument to a `string` array with the same dimensions and sizes. For more information, see [Handle Data Returned from Java Methods](#).

## Python: Supports string data type

MATLAB provides the following string data type functionality when using Python features. For more information about string data types, see [Characters and Strings](#).

- When calling a Python function, MATLAB converts `string` scalar arguments to `py.str`. MATLAB converts `<missing>` values in `string` arguments to `py.None`. For more information, see [Pass Data to Python](#).

- The `string` function converts `py.str` and `py.unicode` types to `string` scalar. For more information, see `Handle Data Returned from Python`. The `string` function does not convert objects defining `__str__` methods; for example, `py.list` or `py.dict` objects.
- The functions `pyversion` and `pyargs` accept the `string` data type for input arguments.
- `string` scalar arguments can be used to index into Python mapping types. For example, for the `py.dict` object `patient = py.dict(pyargs('name', 'John Doe', 'billing', 127));`, you can display the billing value with the statement `patient{"billing"}`.

## Python Version 3.3: Support discontinued

Support for Python version 3.3 is discontinued.

## Version History

To ensure continued support for your applications, upgrade to a supported version of Python—version 3.4 or 3.5.

## MATLAB ships with ActiveState Perl version 5.24 on Windows platforms

MATLAB ships with an updated version of Perl, version 5.24.

## Version History

If you use the `perl` command on Windows platforms, refer to the ActiveState website, <https://www.activestate.com>, for information about this version.

## Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2017	Windows
Added	Intel Parallel Studio XE 2017 for Fortran	macOS
Added	Xcode 8.x, as of R2016b	macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see `Supported and Compatible Compilers`.

## Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Negation of <code>matlab.unittest.constraints.ReturnsTrue</code> constraint ( <code>~ReturnsTrue</code> )	Still runs	Not applicable	Change the logic for tests that rely on the negating the <code>ReturnsTrue</code> constraint.

Functionality	Result	Use This Instead	Compatibility Considerations
Generating test suites from a class that derives from a concrete base class that defines methods that reference a <code>TestParameter</code> , <code>MethodSetupParameter</code> , or <code>ClassSetupParameter</code> not defined within the base class	Warns	Define either the: <ul style="list-style-type: none"> <li>Base class as Abstract using the class-level Abstract attribute. For example, <code>classdef (Abstract) MyTest &lt; matlab.unittest.TestCase</code>.</li> <li>Abstract parameter properties for all parameters used by methods of the class. For example, <code>properties (Abstract, TestParameter)</code>.</li> </ul>	In future releases, if your test class inherits from a concrete base class that uses a parameter that is not defined in the base class, MATLAB will throw an error.
Changes to <code>fieldnames</code> causes the syntax <code>obj.fieldnames</code> to issue an error with some types of objects.	Errors	Use this syntax instead: <code>fieldnames(obj)</code>  To determine what properties are defined for MATLAB objects, use the <code>properties</code> function.	Use the <code>fieldnames</code> function with objects of type COM and Java.
Defining a property get method for a constant property is not supported.	Warns	Constant properties do not support the use of a get method because the value does not change by design.	In future releases, defining a property get method for a constant property will result in an error.
Linking to HTML anchor elements from custom documentation is not supported	Still runs	Not Applicable	Custom documentation pages containing links to anchors still display properly, but the browser does not scroll to the linked location.
Built-in support for the Microsoft Source Code Control Interface, including <code>cmopts</code> and <code>verctrl</code>	Errors	None	MATLAB no longer includes built-in support for the Microsoft Source Code Control Interface (MSSCCI). Replace this functionality with one of these options: <ul style="list-style-type: none"> <li>Download and install the MATLAB integration with MSSCCI add-on.</li> <li>Use the built-in support for Source Control Integration.</li> </ul>

Functionality	Result	Use This Instead	Compatibility Considerations
checkin checkout customverctrl undocheckout	Errors	None	Replace this functionality with one of these: <ul style="list-style-type: none"><li>• Built-in support for Source Control Integration</li><li>• Source Control Software Development Kit to create a plug-in for your source control</li><li>• MATLAB system function and the command-line API for your source control tool to replicate existing functionality</li></ul>

