Phased Array System Toolbox™ Release Notes

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Phased Array System Toolbox™ Release Notes

Trademarks

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

Patents

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

Contents

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2014a

Version: 2.2

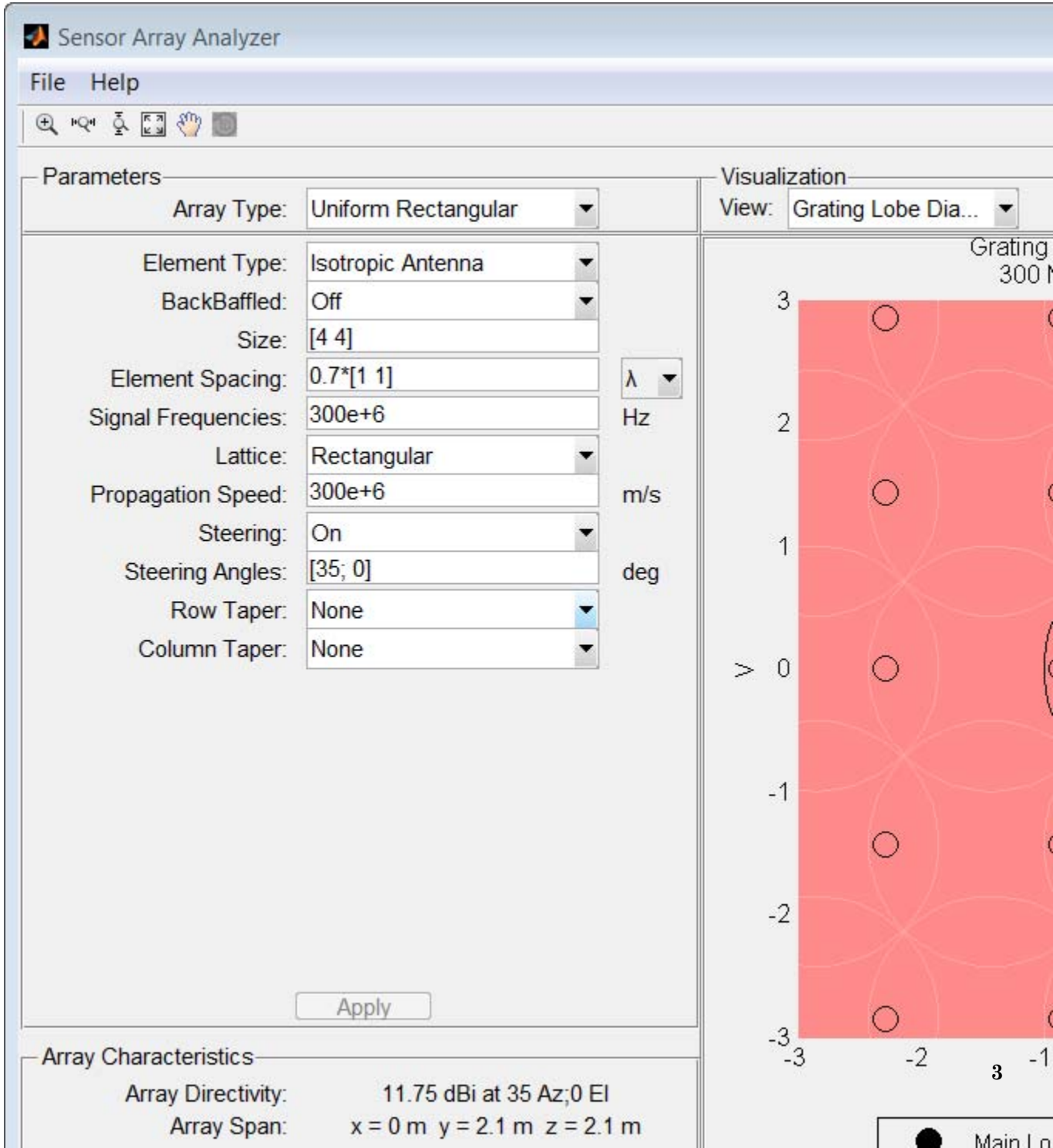
New Features: Yes

Bug Fixes: Yes

Grating lobe diagrams

You can plot the locations of the grating lobes of uniform linear arrays using the new ULA System object™ method `plotGratingLobeDiagram`. For uniform rectangular arrays, you use the URA System object method `plotGratingLobeDiagram`. Using grating lobe diagrams, you can visualize the grating-lobe-free scan region of the array.

The `sensorArrayAnalyzer` app also provides an option for plotting grating lobe diagrams for uniform linear arrays, uniform rectangular arrays, uniform hexagonal arrays, and circular planar arrays. For example, here is the grating lobe diagram of a spatially undersampled 4-by-4 URA steered towards 35° azimuth.


Sensor Array Analyzer
File Help
Parameters
Array Type: Uniform Rectangular
Element Type: Isotropic Antenna
BackBaffled: Off
Size: [4 4]
Element Spacing: 0.7*[1 1] λ
Signal Frequencies: 300e+6 Hz
Lattice: Rectangular
Propagation Speed: 300e+6 m/s
Steering: On
Steering Angles: [35; 0] deg
Row Taper: None
Column Taper: None
Apply
Array Characteristics
Array Directivity: 11.75 dBi at 35 Az;0 El
Array Span: x = 0 m y = 2.1 m z = 2.1 m
Visualization
View: Grating Lobe Dia...
Grating
300 M
3
2
1
0
-1
-2
-3
-3
-2
-1
Main Lo

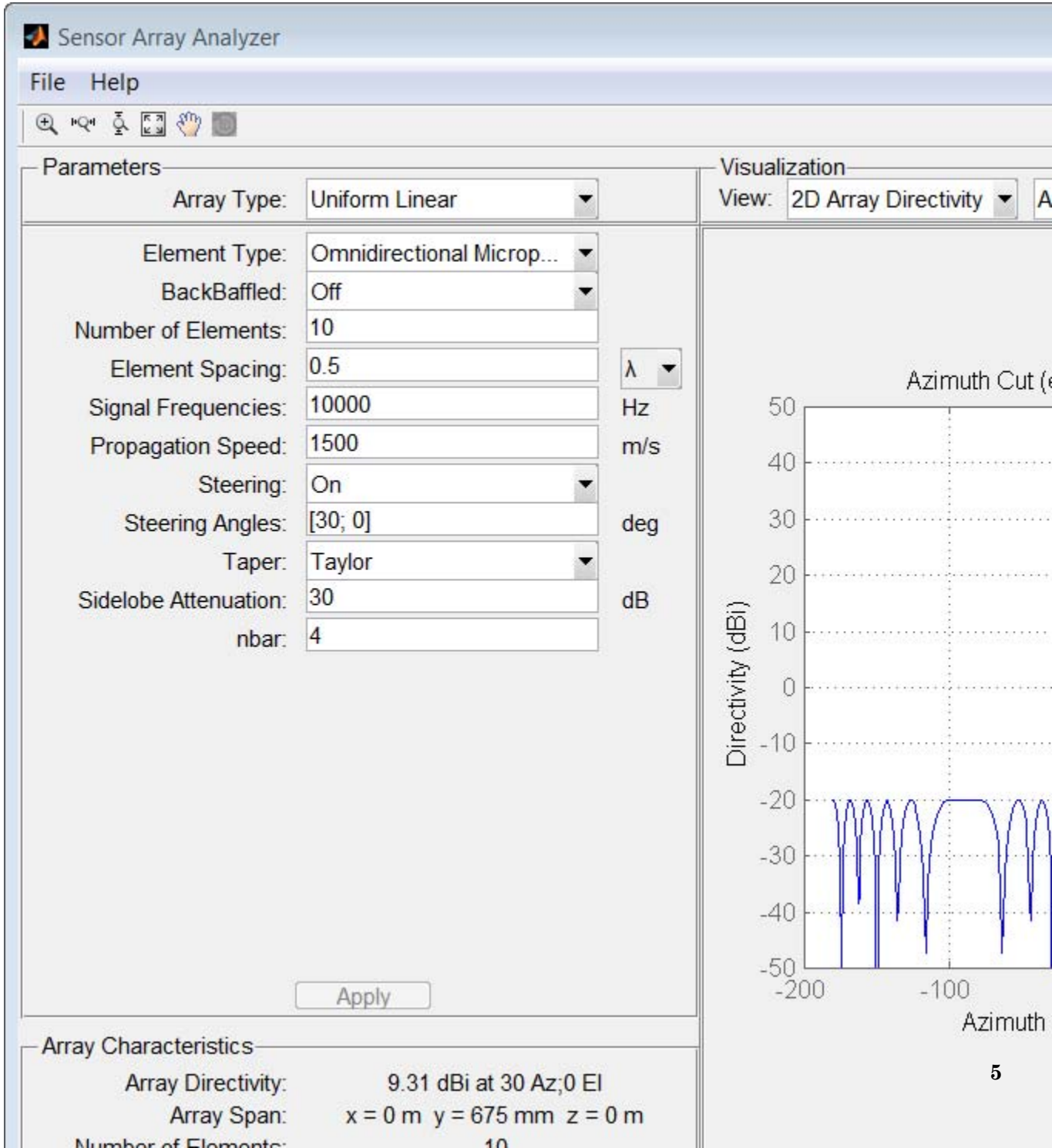
Visualization of element and array directivity

In R2014a, you have a directivity plotting option for all sensor elements, arrays, and subarrays. Directivity patterns help you analyze sensor element and array performance. To plot the directivity, choose `'dbi'` for the `Unit` parameter value of the `plotResponse` method.

The directivity pattern of an array takes into account the directivity pattern of its constituent sensors. Examples of how to create directivity pattern plots for selected sensor elements, arrays and subarrays can be found here.

`Cosine antenna element`
`Uniform line array`
`Partitioned uniform rectangular array`

The `sensorArrayAnalyzer` app also plots 2D and 3D array directivity patterns. You can plot array directivity patterns for all supported array types by choosing `2D Array Directivity` or `3D Array Directivity` from the **Visualization** drop-down list. When you plot 2D directivities, you can choose the azimuth and elevation angles at which the cuts are taken. These plots replace the 2D and 3D array response plots. Here is an example of a directivity pattern plot of a 10-element uniform linear array.


Sensor Array Analyzer
File Help
Parameters
Array Type: Uniform Linear
Element Type: Omnidirectional Microp...
BackBaffled: Off
Number of Elements: 10
Element Spacing: 0.5 λ
Signal Frequencies: 10000 Hz
Propagation Speed: 1500 m/s
Steering: On
Steering Angles: [30; 0] deg
Taper: Taylor
Sidelobe Attenuation: 30 dB
nbar: 4
Apply
Array Characteristics
Array Directivity: 9.31 dBi at 30 Az;0 El
Array Span: x = 0 m y = 675 mm z = 0 m
Visualization
View: 2D Array Directivity
Azimuth Cut (
Directivity (dBi)
50
40
30
20
10
0
-10
-20
-30
-40
-50
-200
-100
Azimuth

Arbitrary geometry and custom element support in the Sensor Array Analyzer app

In R2014a, there are significant modifications and additions to the `sensorArrayAnalyzer` app.

- You can specify any array geometry by choosing `Arbitrary Geometry` from the **Array Type** drop-down list. Then, you specify the element positions in the **Element Position** field and the element normal angles in the **Element Normal** field. You can enter the element positions and normal angles directly or use MATLAB® variables and arrays defined at the command prompt.
- You can define your own antenna type by choosing `Custom Antennas` from the **Element Type** drop-down list. You must supply a radiation pattern in the **Radiation Pattern** field. You must also supply values to the **Frequency Vector**, **Frequency Response**, **Azimuth Angles**, and **Elevation Angles** fields. The azimuth angle dimensions must match the column dimensions of the radiation pattern. The elevation angle dimensions must match the row dimensions of the radiation pattern. You can enter antenna values directly into the app fields or use a MATLAB variable or array defined at the command prompt.
- You can plot array directivity patterns for all supported arrays, as described in "Visualization of element and array directivity" on page 4.
- You can plot grating lobe diagrams for several types of arrays, as described in "Grating lobe diagrams" on page 2.
- You can apply custom taper weights (also known as shading weights) to the array elements for all array geometries. Specify the weights in the **Taper** field directly or use a MATLAB array defined at the command prompt.
- For convenience, you can use MATLAB variables and arrays as entries into some of the app data fields. Instead of typing numerical values into the fields, you can define the values as variables or arrays at the MATLAB command prompt and use the name of the variable or array in the field.

Pulse repetition interval parameter for the Radar Waveform Analyzer app

In R2014a, with the `radarWaveformAnalyzer` app, you can specify the temporal spacing between pulses as either the *pulse repetition frequency*, `PRF`, in hertz, or as the *pulse repetition interval*, `PRI`, in seconds, where `PRI = 1/PRF`.

Property name change in phased.PhaseCodedWaveform

Compatibility Considerations: Yes

The `Type` property of the `phased.PhaseCodedWaveform` System object has been renamed to `Code` property. This System object creates a phase-coded pulse waveform. The `Code` property specifies the code that you use in phase modulation. This change is a name change only. The `Type` property name will be removed in a future release.

Compatibility Considerations

To avoid future incompatibility, change all instances of this property name to the new name.

System object templates

Compatibility Considerations: Yes

The MATLAB® **New > System object** menu now has three new class-definition file templates. The **Basic** template sets up a simple System object. The **Advanced** template includes additional features of System objects. The **Simulink Extension** template provides additional customization of the System object for use in the MATLAB System block.

System objects infer number of inputs and outputs from `stepImpl` method

When you create a new kind of System object that has a fixed number of inputs or outputs specified in the `stepImpl` method, you no longer need to include `getNumInputsImpl` or `getNumOutputsImpl` in your class definition file. The correct number of inputs and outputs are inferred from the `stepImpl` inputs and outputs, respectively.

System objects `infoImpl` method allows variable inputs

When you create a new kind of System object, you can use the `info` method to provide information specific to that object. The `infoImpl` method, which you include in your class-definition file, now allows `varargin` as an input argument.

System objects base class renamed to `matlab.System`

Compatibility Considerations: Yes

The System object base class, `matlab.system.System` has been rename to `matlab.System`. If you use `matlab.system.System` when defining a new System object, a error message results.

Compatibility Considerations

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

System objects Propagates mixin methods

Four new methods have been added to the Propagates mixin class. You use this mixin when creating a new kind of System object for use in the MATLAB System block in Simulink®. You use these methods to query the input and specify the output of a System object.

- `propagatedInputComplexity`

- `propagatedInputDataType`
- `propagatedInputFixedSize`
- `propagatedInputSize`

R2013b

Version: 2.1

New Features: Yes

Bug Fixes: No

Code generation for functions and objects

The Phased Array System Toolbox™ lets you generate C/C++ code from your phased-array MATLAB application code using the MATLAB Coder™. You can create your own standalone C/C++ executables, libraries, and MEX functions directly and automatically from code that you have written that uses phased-array System objects and functions. MATLAB Coder supports basic MATLAB language features such as program control, functions, and matrix operations. See About Code Generation for more information on the use of MATLAB Coder with the Phased Array System Toolbox.

Visualization of element and array radiation patterns with arbitrary resolution

In this release, the `plotResponse` method now lets you plot the radiation pattern with different display ranges and resolutions. Previously, for example, you could only plot the pattern in one degree increments in azimuth and elevation. New display options, `AzimuthAngles`, `ElevationAngles`, `UGrid`, and `VGrid`, give you freedom to change the display range and resolution of the output in azimuth and elevation or in U/V space. These new options apply to all antenna and microphone elements, arrays and subarrays. For documentation and usage examples, see `phased.CosineAntennaElement/plotResponse`, `phased.URA/plotResponse`, and `phased.ReplicatedSubarray/plotResponse`.

Plot response of multiple sets of weights for a single frequency

The `Weights` display option of the `plotResponse` method has a new feature. This applies to all array, replicated subarray and partitioned array System objects. Previously, you could only specify one weight set for multiple frequencies or a different weight set for each frequency. You can now display the response due to multiple sets of weights for a single frequency. For documentation and examples of how to use this properties, see `phased.URA/plotResponse`, and `phased.ReplicatedSubarray/plotResponse`.

New default frequency limits for antenna and microphone elements

The restriction on the range of valid frequencies for all antenna and microphone elements is effectively eliminated by increasing the maximum frequency value to 10^{20} and setting the minimum value to zero.

Function `delayseq` implements FFT length of power of two

Compatibility Considerations: Yes

The function `delayseq` now use an internal *FFT* length equal to a power of two.

Compatibility Considerations

This change in internal logic produces some insignificant numerical differences. No action is required.

System objects matlab.system.System warnings

Compatibility Considerations: Yes

The System object base class, `matlab.system.System` has been replaced by `matlab.System`. If you use `matlab.system.System` when defining a new System object, a warning message results.

Compatibility Considerations

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

Restrictions on modifying properties in System object Impl methods

Compatibility Considerations: Yes

When defining a new System object, certain restrictions affect your ability to modify a property.

You cannot use any of the following methods to modify the properties of an object:

- `cloneImpl`
- `getDiscreteStateImpl`
- `getNumInputsImpl`
- `getNumOutputsImpl`
- `validateInputsImpl`
- `validatePropertiesImpl`

This restriction is required by code generation, which assumes that these methods do not change any property values. These methods are validation and querying methods that are expected to be constant and should not impact the algorithm behavior.

Compatibility Considerations

If any of your class definition files contain code that changes a property in one of the above `Impl` methods, move that property code into an allowable `Impl` method. Refer to the System object `Impl` method reference pages for more information.

plotResponse method handle graphics property change

Compatibility Considerations: Yes

When you use the `plotResponse` method to return a 3D display of the element or array response in *UV* coordinates, the sample coordinates of the displayed data have changed. The data contained in the handle graphics properties, `` `XData' ``, `` `YData' ``, and `` `ZData' ``, are now sampled uniformly in Cartesian *UV* grid coordinates instead of uniformly in polar grid coordinates. This changed

applies to the `plotResponse` method for all antenna and microphone element System objects and array System objects.

Compatibility Considerations

The data is now in uniformly-sampled Cartesian *UV* coordinates. The `plotResponse` display appearance remains unchanged.

Single normal vector for conformal arrays

When you use a conformal array System object, you can specify a single normal direction vector for the case when all the element normal vectors point in the same direction. Previously, the `ElementNormal` property required you to specify a separate normal direction vector for each element. This syntax applies to the `phased.ConformalArray` and `phased.HeterogeneousConformalArray` System objects.

R2013a

Version: 2.0

New Features: Yes

Bug Fixes: Yes

Polarization support for antennas, arrays, and targets that includes transmission, propagation, and reception of polarized signals

A major enhancement to the Phased Array System Toolbox product for R2013a lets you simulate the transmission, propagation and reception of polarized electromagnetic waves. Polarization simulation is turned on by setting a new property `EnablePolarization` in `phased.Radiator`, `phased.Collector`, `phased.WidebandCollector`, `phased.RadarTarget`, `phased.SteeringVector` and `phased.ArrayResponse` System objects. Two new types of antennas specifically for polarized waves are introduced:

- `phased.ShortDipoleAntennaElement` models a short dipole antenna element.
- `phased.CrossedDipoleAntennaElement` models a crossed-dipole antenna element.

You can test whether an antenna or array of antennas can be used to simulate polarization by invoking the `isPolarizationCapable` method. Only `phased.ShortDipoleAntennaElement`, `phased.CrossedDipoleAntennaElement`, and `phased.CustomAntennaElement` support polarization. Polarization properties of arrays depend upon the properties of their constituent antenna elements.

The `phased.CustomAntennaElement` System object has several new properties for polarization. You can use the `SpecifyPolarizationPattern` property to specify whether to use a horizontal or vertical radiation pattern or a combined pattern and then specify `HorizontalMagnitudePattern`, `HorizontalPhasePattern`, `VerticalMagnitudePattern`, and/or `VerticalPhasePattern` to create the pattern itself.

Changes have been made to the `step` and `plotResponse` methods for antenna elements and arrays. With polarization enabled, the `step` method returns a `struct` instead of a data array. The `plotResponse` method plots the horizontal polarization response, the vertical polarization response or a combined polarization response when you set the name-value property `Polarization` to `H`, `V` or `Combined`. This applies to only arrays and antennas

that are capable of polarization. When an antenna or array is not capable of polarization, a fourth option, `None`, is required.

The `phased.RadarTarget` System object lets you model the response of a target to a polarized field by invoking the `EnablePolarization` property. The new `Mode` property allows for `monostatic` or `bistatic` antenna-target configurations. You can set the target's complex 2-by-2 radar cross-section matrix using the new `ScatteringMatrix` property. The scattering matrix contains the *HH*, *HV*, *VH*, and *VV* responses of the target.

Summary of Phased Array Polarization Capabilities

Category	System Objects	New and Modified Properties	New and Modified Methods
Antennas and Microphone Elements	phased.CosineAntennaElement phased.CustomMicrophoneElement phased.IsotropicAntennaElement phased.OmnidirectionalMicrophoneElement phased.CrossedDipoleAntennaElement (new) phased.ShortDipoleAntennaElement (new)		isPolarizationCapable plotResponse step
	phased.CustomAntennaElement	SpecifyPolarizationPattern HorizontalMagnitudePattern HorizontalPhasePattern VerticalMagnitudePattern VerticalPhasePattern	
Array Geometries and Analysis	phased.ConformalArray phased.ULA phased.URA phased.PartitionedArray phased.ReplicatedSubarray		isPolarizationCapable plotResponse step
	phased.SteeringVector phased.ArrayResponse	EnablePolarization	

Category	System Objects	New and Modified Properties	New and Modified Methods
Signal Radiation and Collection	phased.Collector phased.WidebandCollector phased.Radiator	EnablePolarization	step
Environment and Target Models	phased.RadarTarget	EnablePolarization Mode MeanRCSMatrix	step

To support polarization simulation analysis, this release includes new utility functions:

Utility Functions

Name	Description
`polellip`	Parameters of ellipse traced out by tip of a polarized field vector
`polratio`	Ratio of vertical to horizontal linear polarization components of a field
`stokes`	Stokes parameters of polarized field
`circpol2pol`	Convert circular component representation of field to linear component representation
`pol2circpol`	Convert linear component representation of field to circular component representation
`polloss`	Polarization loss
`polsignature`	Radar cross section polarization signature
`rotx`	Rotation matrix for rotations around *x*-axis
`roty`	Rotation matrix for rotations around *y*-axis
`rotz`	Rotation matrix for rotations around *z*-axis
`sph2cartvec`	Convert vector from spherical basis components to Cartesian components

Name	Description
`cart2sphvec`	Convert vector from Cartesian components to spherical representation
`azelaxes`	Spherical basis vectors in 3-by-3 matrix form

Array tapers for the modeling of magnitude and phase perturbations

This release adds element taper (array weighting) support to `phased.ULA`, `phased.URA` , and `phased.ConformalArray` System objects using the new `Taper` property. Tapers can be complex-valued coefficients. Real-valued tapers are usually used, for example, to reduce sidelobe levels; complex tapers are useful for modeling phase perturbations. A new method, `getTaper`, lets you retrieve the taper values. The method, `viewArray`, has been modified to display the array with taper values shown.

Arrays with multiple element patterns, enabling the modeling of edge effects and pattern perturbations

Sensor arrays can now be created to have different antenna patterns assigned to different sensors. These are called *heterogeneous arrays*. To enable this capability, three new system objects are being introduced, `phased.HeterogeneousULA`, `phased.HeterogeneousURA`, and `phased.HeterogeneousConformalArray`. Heterogeneous arrays let you model, for example, cross-element coupling effects or pattern perturbations. You can specify the types of elements you want in the array using the `ElementSet` property and then assign a type to each sensor location using the `ElementIndices` property.

Radar Equation Calculator, Radar Waveform Analyzer, and Sensor Array Analyzer apps

R2013a introduces the first three Phased Array System Toolbox apps:

- `radarEquationCalculator` starts the Radar Equation Calculator which lets you solve for any one of *Target Range*, *Peak Transmit Power,* or *SNR* from the well-known radar equation. Alternatively, you can start Radar Equation Calculator by selecting it from the SIGNAL PROCESSING AND COMMUNICATIONS section of the Apps tab in the MATLAB toolstrip.
- `radarWaveformAnalyzer` invokes the Radar Waveform Analyzer which can plot the shape of five common waveforms as well as their spectra and ambiguity functions. Alternatively, you can start the Radar Waveform Analyzer app by selecting it from the SIGNAL PROCESSING AND COMMUNICATIONS section of the Apps tab in the MATLAB toolstrip.
- `sensorArrayAnalyzer` starts Sensor Array Analyzer to display the geometry of eight different common array configurations. It can also plot the 2-D and 3-D array responses. The user can set the number of array elements, element types, and element spacings and other parameters. Alternatively, you can start the Sensor Array Analyzer app by selecting it from the SIGNAL PROCESSING AND COMMUNICATIONS section of the Apps tab in the MATLAB toolstrip.

Visualization of radar vertical coverage on Blake charts

New radar analysis tools, `blakechart` and `radarvcd`, plot radar range-height-angle (Blake) charts and vertical coverage diagrams. You can use these radar design tools to predict the maximum radar range. This function employs the CRPL Exponential Reference Atmosphere model of the refractive index of the atmosphere.

Custom antenna element patterns specified at different frequencies

This new feature of `phased.CustomAntennaElement` lets you specify frequency-dependent antenna patterns. You do so by creating a 3D array containing pattern values for azimuth, elevation and frequency.

Full GPU support for clutter model, including nonisotropic antennas and subarrays (using Parallel Computing Toolbox)

You can now use the `phased.gpu.ConstantGammaClutter` System object to accelerate clutter simulations with all antenna types not just isotropic. This System object typically runs faster than `phased.ConstantGammaClutter`. However, there is no GPU support for clutter modeling of polarized waves. `phased.gpu.ConstantGammaClutter` requires a license for the Parallel Computing Toolbox™ and a GPU-enabled computer.

Ordinary functions for narrowband beamformers

These new functions give you tools to compute narrowband beamformer weights without requiring the system object framework:

- `steervec` computes the steering vector for a narrowband conventional 1D, , or 3D beamformer of arbitrary shape. The antenna elements are assumed to be isotropic. Inputs to this function are the element positions in units of wavelength and the directions-of-arrival of the incoming signals.
- `cbfweights` computes the weights for a narrowband conventional 1D, , or 3D beamformer of arbitrary shape. The antenna elements are assumed to be isotropic. Inputs to this function are the element positions in units of wavelength and the directions-of-arrival of the incoming signals. The weights produced by `cbfweights` equal those produced by `steervec` divided by the number of elements in the array.
- `mvdrweights` returns the weights for a narrowband minimum variance distortionless response (MVDR) beamformer. Inputs to this function are the element positions in units of wavelength, the directions-of-arrival of the incoming signals, and the sensor covariance matrix.
- `lcmvweights` returns the weights for a narrowband linear constraint minimum variance (LCMV) beamformer. Inputs are the signal covariance matrix, the desired responses, and the constraint matrix.
- `sensorcov` returns the received spatial covariance matrix for narrowband plane wave signals arriving at a sensor array. Inputs are the sensor element positions in units of wavelength and the directions-of-arrival of

the incoming signals. Optional inputs are the sensor noise and signal covariance matrices.

Ordinary functions for narrowband signal directions-of-arrival at a uniform line array

These new functions allow you to compute directions-of-arrival (DOA) of narrowband signals for uniform line arrays without requiring the use of the system object framework.

- `rootmusicdoa` computes, using the Root MUSIC algorithm, a vector of estimated arrival directions of multiple signals. This estimator uses the sensor covariance matrix and requires that the number of signals be a known value.
- `espritdoa` computes, using the TLS ESPRIT algorithm, a vector of estimated arrival directions of multiple signals. This estimator uses the sensor covariance matrix and requires that the number of signals be a known value.
- `aictest` estimates the number of signals arriving at an array using the Akaike Information Criterion test. This estimator uses a set of snapshots taken at each sensor.
- `mdltest` estimates the number of signals arriving at an array using the Minimum Description Length test. This estimator uses a set of snapshots taken at each sensor.
- `spsmooth` performs spatial smoothing (averaging) of a covariance matrix using maximum overlapped subarrays.

Deprecated `VisibleRegion` in `phased.ESPRITEstimator`

Compatibility Considerations: Yes

The `VisibleRegion` property of the `phased.ESPRITEstimator` System object will be removed in a future release.

Compatibility Considerations

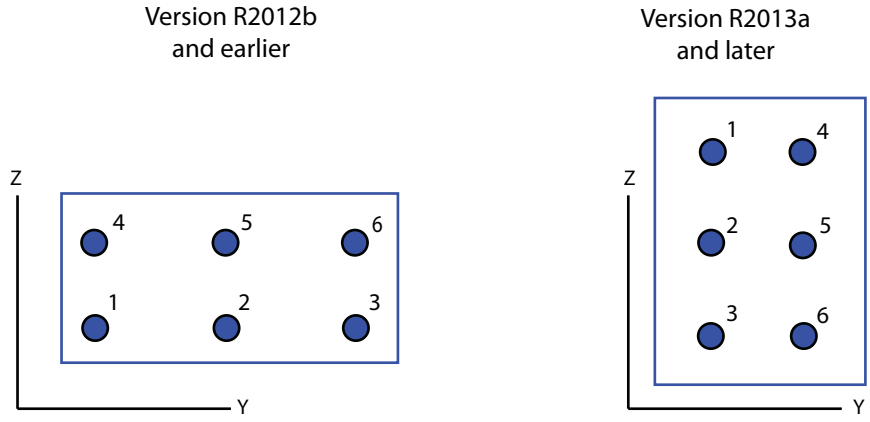
In the future, users will have to remove the use of this property from their code.

Element indexing pattern change for `phased.URA` and `phased.ReplicatedSubarray`

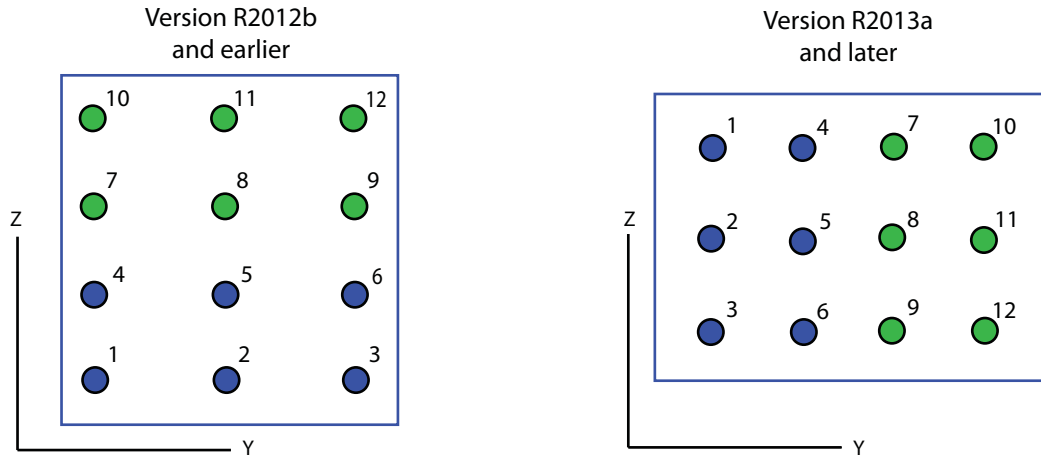
Compatibility Considerations: Yes

The index order of the elements of a uniform rectangular array as constructed in `phased.URA` has changed with this release. Instead of row-major order, elements are stored in column-major order. This has implications for the size and shape of an array. The size of the array is still specified by the `Size` property which is a 1-by-2 integer vector (or a single integer for square arrays). This vector is now interpreted as `[NumberOfRows, NumberOfColumns]`. The corresponding `ElementSpacing` property is a 1-by-2 vector containing the distance between elements (in meters) as `[SpacingBetweenRows, SpacingBetweenColumns]`. If `ElementSpacing` is a scalar, the distance along the columns and rows is the same. The following figure shows how the indexing and shape of an array are changed with this release by using a 6-element rectangular array as an example. Previously, a [3,2] array would have three columns and two rows; now it has two columns and three rows.

New Size and Element Indexing Order for Uniform Rectangular Arrays
Example: Size = [3,2]



When constructing replicated subarrays using `phased.ReplicatedSubarray`, the `GridSize` and `GridSpacing` properties are used to position the subarrays on a rectangular grid. You specify the dimensions of the grid using the `GridSize` property and the distance between grid points with `GridSpacing` property. As with `phased.URA`, the interpretation of these properties has changed with this release. In this release, `GridSize` is a 1-by-2 vector in the form of `[NumberOfRows NumberOfColumns]` gives the number of elements along each column (z-axis) and the number of elements along each row (y-axis). If `GridSize` is a scalar, the replicated array has the same number of grid points in each row and column. The `GridSpacing` property can either be 1-by-2 vector in the form of `[SpacingBetweenRows SpacingBetweenColumns]` or a scalar (units are meters). If `GridSpacing` is a scalar, the distance along the rows and columns is the same. The following figure shows how the indexing and shape of a replicated subarray are changed with this release using a `GridSize` of `[1,2]`. This creates a 1-by-2 grid of subarrays as shown on the right-hand side of the figure.



These changes make antenna array indexing consistent with the MATLAB convention. To insure that the user is aware of this important change, a warning message will be displayed when `phased.URA` are invoked. This message may be suppressed by typing `warning off phased:system:array:SizeConventionWarning` at the command line at any time or by including it in your `startup.m` script and other scripts. A similar warning appears when `phased.ReplicatedSubarray` is invoked. Type `warning off phased:system:array:GridConventionWarning` to suppress this message.

Compatibility Considerations

Since data is now stored differently, results will generally differ from that of previous releases for the same input. In most cases however, interchanging the order of entries in the `Size` and `ElementSpacing` vectors for `phased.URA` and the `GridSize` and `GridSpacing`vectors for `phased.ReplicatedSubarray`

will give the same results. If you need to look at the output of individual array elements, then there will be some differences.

MATLAB Compiler Support

Phased Array System Toolbox supports the MATLAB Compiler™ for all functions and System objects. Compiler support does not extend to any of the toolbox apps.

New method for action when System object input size changes

The `processInputSizeChangeImpl` method allows you to specify actions to take when an input to a System object you have defined changes size. If an input changes size after the first call to `step`, the actions defined in `processInputSizeChangeImpl` occur when `step` is next called on that object

R2012b

Version: 1.3

New Features: Yes

Bug Fixes: No

Acceleration of clutter model simulation with `parfor` or GPUs

A change in the `phased.ConstantGammaClutter` System object facilitates performing Monte Carlo simulations with Parallel Computing Toolbox constructs, such as `parfor`. For details about this change, see the description of random number stream usage that follows.

The new `phased.gpu.ConstantGammaClutter` System object simulates clutter on a GPU. This System object typically runs faster than `phased.ConstantGammaClutter`. Using `phased.gpu.ConstantGammaClutter` requires a license for Parallel Computing Toolbox software.

FMCW waveforms

The `phased.FMCWWaveform` System object models a frequency modulated continuous wave (FMCW) waveform.

These functions help you determine appropriate property values for `phased.FMCWWaveform`:

- `time2range`
- `range2time`
- `range2bw`

These functions help you simulate and analyze systems that process FMCW waveforms:

- `dechirp`
- `beat2range`
- `range2beat`
- `rdcoupling`

CFAR detector, supporting SOCA, GOCA, and order statistic thresholds

Compatibility Considerations: Yes

The `phased.CFARDetector` System object provides a new property named `Method`. Using this property, you can choose among four CFAR detection algorithms: cell averaging (default), smallest-of cell averaging (SOCA), greatest-of cell averaging (GOCA), and order statistic.

Compatibility Considerations

In R2012b, if you save a CFAR detector variable in a MAT-file, you cannot load that variable from the MAT-file in an earlier version. Instead, re-create the variable in the earlier version.

Function to simulate signals received by an array

The new `sensorsig` function simulates plane wave signals received at a phased array. This function facilitates statistical analysis and testing of direction-of-arrival algorithms. The function does not require you to simulate an entire phased array system.

Range-Doppler estimation

The new `phased.RangeDopplerResponse` System object generates and plots range-Doppler maps.

Propagation model that now supports intrapulse Doppler modeling

Compatibility Considerations: Yes

The `phased.FreeSpace` System object models the Doppler shift both within a pulse (fast time) and between successive pulses (slow time). In previous releases, the object modeled only the slow-time Doppler shift.

This enhanced Doppler shift modeling is especially useful for observing range-Doppler coupling in a radar system that uses a linear FM waveform.

Compatibility Considerations

The `step` method of the `phased.FreeSpace` System object has two additional input parameters:

- `origin_vel`, the velocity of the signal origin
- `dest_vel`, the velocity of the signal destination

To update legacy code that uses this `step` method, use one of these approaches:

- In some cases, the signal origin or signal destination is stationary. If so, set the corresponding velocity input argument to `[0; 0; 0]`.
- In other cases, the signal origin or signal destination is moving, and you are using `phased.Platform` to model the moving platform. In this situation, obtain the velocity vector as an additional output argument from the `step` method of `phased.Platform`. Then, specify this velocity vector as an input argument in the `step` method of `phased.FreeSpace`. For example, compare the R2012a and R2012b versions of an example, and notice the introduction of the `txvel` variable in the R2012b version.

Visualization of array geometries

You can call `viewArray` on a phased array to plot the positions, normal directions, and element indices of the elements in the array. For arrays containing subarrays, `viewArray` can also graphically highlight one or more of the subarrays.

The System objects affected are:

- `phased.ULA`
- `phased.URA`
- `phased.ConformalArray`
- `phased.ReplicatedSubarray`
- `phased.PartitionedArray`

For more information, see the reference pages for:

- `viewArray` for arrays without subarrays, such as `phased.ULA`
- `viewArray` for arrays containing subarrays, such as `phased.PartitionedArray`

Random number stream usage for parallel computing

Compatibility Considerations: Yes

System objects in Phased Array System Toolbox software that rely on a random number generator now behave differently when you set the `SeedSource` property to `'Auto'`:

- The object uses the global stream of random numbers instead of a private stream. This change is useful if you are performing the computations in a set of Monte Carlo trials involving Parallel Computing Toolbox software. In that situation, the global stream is more suitable than a stream that the System object manages internally.
- The `reset` method does not reset the random number generator.

The System objects affected by this change are:

- `phased.BarrageJammer`
- `phased.ConstantGammaClutter`
- `phased.ReceiverPreamp`
- `phased.ReceiverPreamp`
- `phased.Transmitter`

Compatibility Considerations

The following compatibility considerations apply to System objects that existed in earlier releases, if your legacy code configures the objects with the `SeedSource` property set to `'Auto'`.

- In operations involving the System objects, the specific random numbers in R2012b differ from the random numbers in earlier releases.

- If your code later performs an arbitrary operation that uses the global random number stream, this operation also uses different random numbers compared to earlier releases.
- In some cases, your code may rely on operations in earlier releases that reset or restore the random number generator while loading, cloning, or resetting objects. If so, you should update your code to reset or restore the global stream yourself.

`save` and `load` for System objects

You can create your own `save` and `load` methods for a System object you create. To do so, use the `saveObjectImpl` and `loadObjectImpl`, respectively, in your class definition file.

R2012a

Version: 1.2

New Features: Yes

Bug Fixes: No

Replicated Subarrays and Partitioned Array Apertures

Compatibility Considerations: Yes

A *subarray* is an accessible subset of array elements. The following new System objects enable you to create arrays that contain subarrays:

- `phased.ReplicatedSubarray`
- `phased.PartitionedArray`

The following table lists existing System objects that now support operations on arrays that contain subarrays. Each of the objects in the table has a property called `SensorArray` or `Sensor`. You can set that property to an array object that contains subarrays. Also, some of the objects in the table support subarray steering through a new input argument, `STEERANGLE`, in the `step` method.

System Object	SensorArray or Sensor Can Contain Subarrays	step Syntax Can Include Subarray Steering
`phased.AngleDopplerResponse`	Yes	Not applicable
`phased.ArrayGain`	Yes	Yes
`phased.ArrayResponse`	Yes	Yes
`phased.Collector`	Yes	Yes
`phased.ConstantGammaClutter`	Yes	Yes
`phased.MVDRBeamformer`	Yes	Not applicable
`phased.PhaseShiftBeamformer`	Yes	Not applicable
`phased.Radiator`	Yes	Yes
`phased.STAPSMIBeamformer`	Yes	Not applicable
`phased.SteeringVector`	Yes	Yes
`phased.SubbandPhaseShiftBeamformer`	Yes	Not applicable
`phased.WidebandCollector`	Yes	Yes

For more information about using subarrays, see Subarrays Within Arrays or Subarrays in Phased Array Antennas.

Compatibility Considerations

The `IncludeElementResponse` property of the `phased.SteeringVector` System object is no longer tunable in V1.2 (R2012a). This change facilitates support for arrays containing subarrays.

You may have code from an earlier version that tunes the value of the `IncludeElementResponse` property of a locked steering vector object. If so, the code will produce an error message in R2012a. You can avoid the error message by calling `release` to unlock the object, or by not changing the value of the `IncludeElementResponse` property.

Stretch Processing

The following new features help you perform pulse compression on linear frequency modulation (FM) waveforms using stretch processing:

- `phased.StretchProcessor` System object
- `stretchfreq2rng` function
- `getStretchProcessor` method of `phased.LinearFMWaveform` System object

Stretch processing is sometimes called *deramping* or *dechirping*.

For more information about using stretch processing, see Stretch Processing or Range Estimation Using Stretch Processing.

U/V Space and Phi/Theta Angles

Several enhancements facilitate performing operations in the *u*/*v* coordinate system or in a spherical coordinate system that describes angles using φ and θ instead of azimuth and elevation:

- Visualize radiation patterns in *u*/*v* space using the `plotResponse` method for arrays, antenna elements, and microphone elements. To use this feature, include `'Format','UV'` in the `plotResponse` syntax.
- Convert coordinates from one coordinate system to another using these new functions:
  - `uv2azel`
  - `azel2uv`
  - `phitheta2azel`
  - `azel2phitheta`
  - `uv2phitheta`
  - `phitheta2uv`
- Convert antenna radiation patterns from one coordinate system to another using these new functions:
  - `uv2azelpat`
  - `azel2uvpat`
  - `phitheta2azelpat`
  - `azel2phithetapat`
  - `uv2phithetapat`
  - `phitheta2uvpat`

For examples, see Antenna Radiation Pattern in U/V Coordinates and Antenna Array Analysis with Custom Radiation Pattern. For background information about the coordinate systems, see Spherical Coordinates.

Multiple-Beam Beamformers

Compatibility Considerations: Yes

The following beamformers support multiple beamforming directions:

- `phased.PhaseShiftBeamformer`
- `phased.SubbandPhaseShiftBeamformer`

- `phased.MVDRBeamformer`

You can use this capability to model switched-beam systems.

To indicate multiple beamforming directions, use a matrix instead of a vector for the `Direction` property of the beamformer object or the `ANG` input argument of `step`. In earlier versions, the value required a vector. Now, when you specify multiple beamforming directions, the `Y` and `W` outputs of `step` have an extra matrix dimension.

Compatibility Considerations

In V1.2 (R2012a), you can create a MAT-file that stores a beamformer variable specifying multiple directions in the `Direction` property. However, you cannot load that variable from the MAT-file in an earlier version. As an alternative, you can re-create the variable in the earlier version and specify only one beamforming direction.

Support for Wideband Beam Pattern Analysis

The `plotResponse` method for arrays, antenna elements, and microphone elements has enhancements for use with wideband beamforming applications.

- For arrays or elements, `plotResponse` can plot multiple frequency responses in a three-dimensional waterfall plot. To use this feature, include `'OverlayFreq',false` in the `plotResponse` syntax. The `OverlayFreq` argument is new.
- For arrays, `plotResponse` can apply weights independently to each frequency in the plot. For example, you can use beamformer weights as in Visualization of Wideband Beamformer Performance. To use this feature, include `'Weights',Value` in the `plotResponse` syntax, where `Value` is a vector or matrix. R2011b required that the weights be the same for all frequencies in the plot and that `Value` be a vector.

In the `phased.ArrayResponse` and `phased.ArrayGain` System objects, the `step` method permits the `WEIGHTS` input argument to be a vector or a matrix. In earlier releases, `WEIGHTS` is a vector.

Beamformer Option to Preserve Power

Compatibility Considerations: Yes

The phase shift beamformer offers options for normalizing the beamformer weights. To select an option, set the new `WeightsNormalization` property of the `phased.PhaseShiftBeamformer` object to one of these values:

- `'Distortionless'` — The gain toward the beamforming direction is 0 dB. This choice is the default and matches the behavior in earlier versions.
- `'Preserve power'` — The norm of the weights is 1.

Compatibility Considerations

In V1.2 (R2012a), if you save a phase shift beamformer variable in a MAT-file, you cannot load that variable from the MAT-file in an earlier version. Instead, re-create the variable in the earlier version.

Symmetric Sweeping of Linear FM Waveform

You can create a linear FM waveform to sweep in an interval that is symmetric about 0 or positive only. To choose the location of the FM sweep interval, set the new `SweepInterval` property of the `phased.LinearFMWaveform` object to one of these values:

- `'Positive'` — The waveform sweeps between 0 and B, where B is the sweep bandwidth. This choice is the default and matches the behavior in earlier versions.
- `'Symmetric'` — The waveform sweeps between $-B/2$ and $B/2$.

Toolbox Location of dutycycle Function

The `dutycycle` function in the Signal Processing Toolbox™ product replaces the earlier `dutycycle` function in the Phased Array System Toolbox product. The new function includes both the capabilities of the earlier function and additional new capabilities.

New System Object Option on File Menu

The File menu on the MATLAB desktop now includes a **New > System object** menu item. This option opens a System object class template, which you can use to define a System object class.

Variable-Size Input Support for System Objects

System objects that you define now support inputs that change size at runtime.

Data Type Support for System Objects

System objects that you define now support all MATLAB data types as inputs and outputs.

New Property Attribute to Define States

R2012a adds the new `DiscreteState` attribute for properties in your System object class definition file. Discrete states are values calculated during one step of an object's algorithm that are needed during future steps.

New Methods to Validate Properties and Get States from System Objects

The following methods have been added:

- `validateProperties` – Checks that the System object is in a valid configuration. This applies only to objects that have a defined `validatePropertiesImpl` method
- `getDiscreteState` – Returns a `struct` containing a System object's properties that have the `DiscreteState` attribute

matlab.system.System changed to matlab.System

Compatibility Considerations: Yes

The base System object class name has changed from `matlab.system.System` to `matlab.System`.

Compatibility Considerations

The previous `matlab.system.System` class will remain valid for existing System objects. When you define new System objects, your class file should inherit from the `matlab.System` class.

R2011b

Version: 1.1

New Features: Yes

Bug Fixes: No

Constant Gamma Clutter Modeling

The new `phased.ConstantGammaClutter` System object helps you model surface clutter using the constant gamma model. You can use this object when simulating a radar system or estimating its performance statistically.

For more information, see these resources:

- Clutter Modeling
- `phased.ConstantGammaClutter`
- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar

Clutter Modeling Utilities

These new utility functions can help you implement custom clutter models:

- `billingsleyicm`
- `depressionang`
- `effearthradius`
- `grazingang`
- `horizonrange`
- `surfclutterrcs`
- `surfacegamma`

Phase-Coded Waveforms

The new `phased.PhaseCodedWaveform` System object generates samples of a phase-coded pulse waveform. This object supports these code types:

- Barker
- Frank
- P1
- P2

- P3
- P4
- Px
- Zadoff-Chu

For more information, see Phase-Coded Waveforms and `phased.PhaseCodedWaveform`.

Spectrum Weighting Options in Matched Filter

Compatibility Considerations: Yes

The `phased.MatchedFilter` System object supports spectrum weighting using these window types:

- Hamming
- Chebyshev
- Hann
- Kaiser
- Taylor

You can also specify a custom window. To do so, write a function that takes the window length as an input argument and returns window coefficients in an output argument.

For more information, see Matched Filtering or `phased.MatchedFilter`.

Compatibility Considerations

If you save a `phased.MatchedFilter` object in a MAT-file in V1.1 (R2011b) and then load the MAT-file in V1.0 (R2011a), the object does not perform spectrum weighting. The Command Window shows this warning:

```
Warning: While loading an object of class 'phased.MatchedFilter':
No public field SpectrumWindow exists for class phased.MatchedFilter.
```

If you write code in V1.1 (R2011b) that sets or reads any of the following properties of `phased.MatchedFilter` object, the code produces an error message in V1.0 (R2011a).

- `SpectrumWindow`
- `CustomSpectrumWindow`
- `SpectrumRange`
- `SampleRate`
- `SidelobeAttenuation`
- `Beta`
- `Nbar`

Expanded Lattice Options in Uniform Rectangular Array

Compatibility Considerations: Yes

The `phased.URA` System object supports both triangular lattices and rectangular lattices. You use the `Lattice` property to select the lattice type.

In V1.0 (R2011a), `phased.URA` supports only rectangular lattices and does not have a `Lattice` property.

Compatibility Considerations

If you save a `phased.URA` object in a MAT-file in V1.1 (R2011b) and then load the MAT-file in V1.0 (R2011a), the object uses a rectangular lattice. The Command Window shows this warning:

```
Warning: While loading an object of class 'phased.URA':
No public field Lattice exists for class phased.URA.
```

If you write code in V1.1 (R2011b) that sets or reads the `Lattice` property of a `phased.URA` object, the code produces an error message in V1.0 (R2011a).

Enhanced Plots Show Multiple Frequency Responses

The `plotResponse` method can plot multiple frequency responses along an azimuth cut or elevation cut. This method is available for the System objects for array design, antenna elements, and microphone elements. To create a plot of multiple frequency responses, use a `plotResponse` syntax in which:

- `FREQ` is a row vector.
- `RespCut` either does not appear explicitly, or has the value `'Az'` or `'El`.

The affected System objects are:

- `phased.ConformalArray`
- `phased.CosineAntennaElement`
- `phased.CustomAntennaElement`
- `phased.CustomMicrophoneElement`
- `phased.IsotropicAntennaElement`
- `phased.OmnidirectionalMicrophoneElement`
- `phased.ULA`
- `phased.URA`

In V1.0 (R2011a), `FREQ` must be a scalar. The resulting plot shows one frequency response.

Custom Antenna Removes Restriction on Radiation Pattern

The `phased.CustomAntennaElement` System object now permits more general radiation patterns. The main beam of the pattern is no longer required to point to 0 degrees azimuth and 0 degrees elevation.

Storing States When Saving or Cloning Objects

Compatibility Considerations: Yes

The `save` and `clone` operations now store all states of the System objects in the `phased` package. As a result, calling the `step` method on a loaded or cloned object resumes processing from the state where the original object left off. In V1.0 (R2011a), the loaded or cloned object is unlocked and uninitialized.

Compatibility Considerations

If your legacy code exploits the unlocked, uninitialized state of a loaded or cloned object, you should update the code in V1.1 (R2011b). You can use the `release` method to unlock objects.

Custom System Objects

You can now create custom System objects in MATLAB. This capability allows you to define your own System objects for time-based and data-driven algorithms, I/O, and visualizations. The System object API provides a set of implementation and service methods that you incorporate into your code to implement your algorithm. See Define New System Objects in the DSP System Toolbox™ documentation for more information.

Conversion of Error and Warning Message Identifiers

Compatibility Considerations: Yes

For version 1.1 (R2011b), some error and warning message identifiers have changed in Phased Array System Toolbox software.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `'phased:phased:RootWSFEstimator:ZeroSourceNumber'` identifier and the

`'phased:phased:RootMUSICEstimator:ZeroSourceNumber'` identifier have both changed to `'phased:phased:doa:ZeroSourceNumber'`. If your code checks for one of the earlier values, you must update it to check for `'phased:phased:doa:ZeroSourceNumber'` instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

Note Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs warning-free.

R2011a

Version: 1.0

New Features: Yes

Bug Fixes: No

Introducing the Phased Array System Toolbox

Phased Array System Toolbox provides algorithms and tools for the design, simulation, and analysis of phased array signal processing systems. These capabilities are provided as MATLAB functions and MATLAB System objects. The system toolbox includes algorithms for waveform generation, beamforming, direction of arrival estimation, target detection, and space-time adaptive processing. The system toolbox lets you build monostatic, bistatic, and multistatic architectures for a variety of array geometries. You can model these architectures on stationary or moving platforms. Array analysis and visualization tools help you evaluate spatial, spectral, and temporal performance. The system toolbox lets you model an end-to-end phased array system or use individual algorithms to process acquired data.

Features

Key features of Phased Array System Toolbox Version 1.0 include:

- Algorithms available as MATLAB functions and MATLAB System objects
- Monostatic, bistatic, and multistatic phased array system modeling
- Array analysis and 3D visualization; physical array modeling for uniform linear arrays, uniform rectangular arrays, and arbitrary conformal arrays on platforms with motion
- Broadband and narrowband digital beamforming functions, including MVDR/Capon, LCMV, time delay, Frost, time delay LCMV, and subband phase shift
- Space-time adaptive processing algorithms, including displaced phase center array (DPCA), adaptive DPCA, sample matrix inversion (SMI) beamforming, and angle-Doppler response visualization
- Direction of arrival algorithms, including MVDR, ESPRIT, Beamscan, Root MUSIC, and monopulse tracking
- Waveform synthesis functions for pulsed CW, linear FM, stepped FM, and staggered PRF signals, and waveform visualization tools for ambiguity function and matched filter response

- Algorithms for TVG, pulse compression, coherent and non-coherent integration, CFAR processing, plotting ROC curves, and estimating range and Doppler