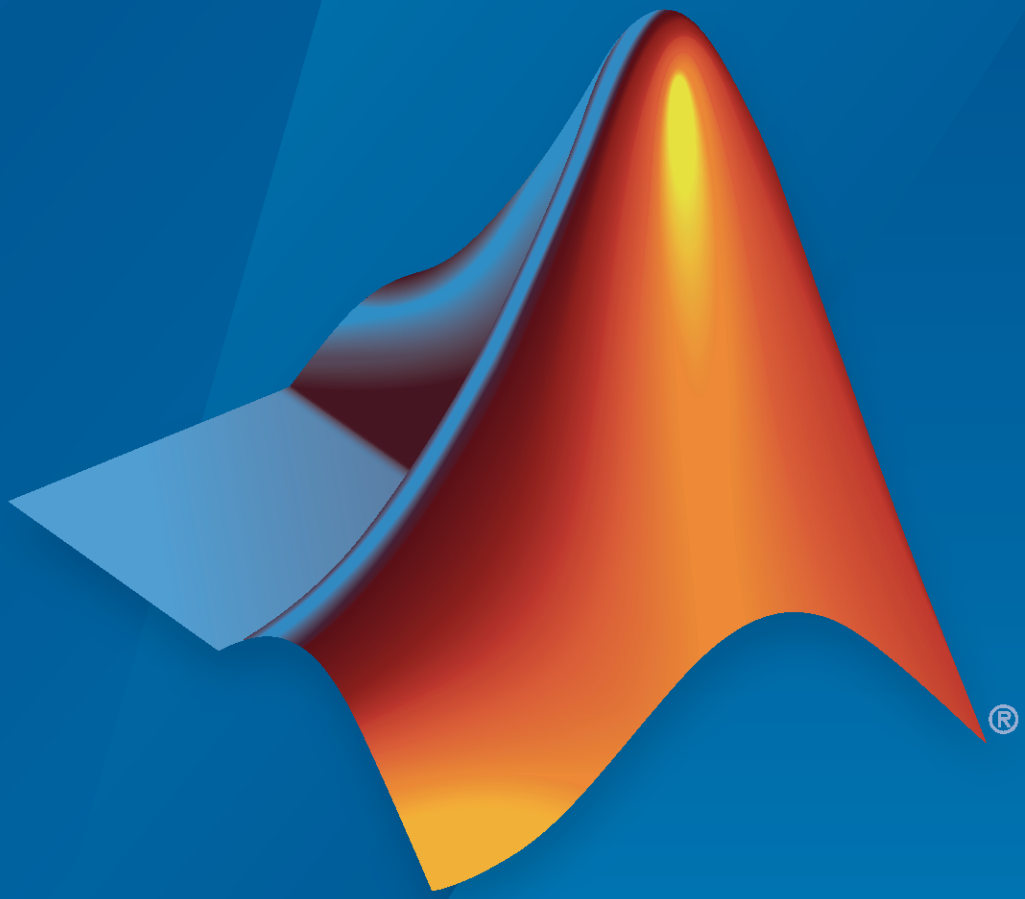
# Phased Array System Toolbox™ Release Notes



MATLAB® & SIMULINK®

MathWorks®

# How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

*Phased Array System Toolbox™ Release Notes*

# Contents

## R2020b



## R2020a



## R2019b

## R2019a



## R2018b

## R2018a



## R2017b

## R2017a



## R2016b



## R2016a

## R2015b

## R2015a



## R2014b

## R2014a



## R2013b

## R2013a



## R2012b

## R2012a



## R2011b

## R2011a

# R2020b

**Version: 4.4**

**New Features**

**Bug Fixes**

## Subarray Support in Sensor Array Analyzer

Starting this release, you can use the **Sensor Array Analyzer** app to analyze arrays containing subarrays. You can create an array with subarrays by replicating one array along a spatial grid. You can also create an array containing subarrays by partitioning a single array into subarrays. You can then display 2-D and 3-D beam patterns and grating lobe diagrams, as well as array performance characteristics.

## Frequency Offset in Pulsed Waveforms

You can now apply frequency shifts to the pulse waveforms produced by the `phased.RectangularWaveform`, `phased.LinearFMWaveform`, `phased.SteppedFMWaveform`, and `phased.PhaseCodedWaveform` System objects. This capability is controlled by the new `FrequencyOffsetSource` and `FrequencyOffset` properties.

Using the `getMatchedFilter` object function, you can obtain the frequency-shifted matched filter coefficients. The `plot` object function shows the frequency-shifted waveforms. Frequency offset is already supported in the `phased.PulseWaveformLibrary` System object™.

This option is also available in the Rectangular Waveform, Linear FM Waveform, Stepped FM Waveform, and Phase Coded Waveform, as well as the Pulse Waveform Library Simulink® blocks.

## Beamwidth of Array and Element Patterns

The new `beamwidth` object function lets you compute and visualize the beamwidth of array or element patterns. You can compute either the half-power beamwidth, the first-null beamwidth, or a custom n-dB down beamwidth for any element, array, or subarray in Phased Array System Toolbox.

## Pulse Compression Library Integration into Radar Waveform Analyzer App

The **Radar Waveform Analyzer** app now supports pulse compression. You can configure and analyze techniques related to matched filtering or stretch processing of the waveforms you select. Export the configurations you design to MATLAB® or Simulink to obtain compression objects and blocks corresponding to the different waveforms.

## Application Examples: HDL Code Generation

This release includes two new examples that employ HDL code generation.

- The “Fixed-Point HDL-Optimized Minimum-Variance Distortionless-Response (MVDR) Beamformer” example shows how to implement a fixed-point HDL-optimized minimum-variance distortionless-response beamformer.
- The “FPGA Based Monopulse Technique Workflow: Design and Code Generation” example uses digital-down conversion and monopulse techniques to estimate the azimuth and elevation of an object.

# R2020a

**Version: 4.3**

**New Features**

**Bug Fixes**

## Multiuser Block Diagonalization Beamforming

The new `blkdiagbfweights` function extends spatial multiplexing from single-user to multi-user MIMO communication systems. The existing `diagbfweights` function computes the precoding and combining weights for a single user. `blkdiagbfweights` computes the precoding and combining weights for multi-user MIMO beamforming using a block diagonalization algorithm.

## Rain Models for Predicting Signal Attenuation

- This release introduces a new model for RF signal attenuation caused by rain. The `cranerainpl` function implements the Global Crane Rain Attenuation model. The Crane model is a more conservative alternative to the ITU model and generally predicts greater losses than the ITU rain attenuation model used in the `rainpl` function.
- A new syntax for the `rainpl` function lets you specify the rainfall exceedance percentage. The function has also been updated to use the newest ITU rain model.

## Updated Effective Earth Radius Model

New syntaxes have been added to the `effearthradius` function. The function returns the effective radius of a spherical earth used to compute RF propagation losses. The new syntaxes let you calculate the effective earth radius using the average radius of curvature method.

## Import Custom Antenna Patterns

You can now import custom antenna patterns expressed in phi-theta coordinates into the `phased.CustomAntennaElement` System object. Previously, you could import patterns specified in terms of azimuth and elevation coordinates. The new `PatternCoordinateSystem` property lets you choose between specifying radiation patterns in azimuth-elevation or phi-theta coordinates. The **Sensor Array Analyzer** app also supports phi-theta coordinates using the **Pattern Coordinate System** parameter when the `Custom Antenna` element is selected. This coordinate system selection option also extends to those Simulink blocks that let you set antenna and array parameters.

## Application Examples: HDL Code Generation

These examples show how you can generate HDL code from Simulink models.

- FPGA Based Beamforming in Simulink: Part 1 - Algorithm Design
- FPGA Based Beamforming in Simulink: Part 2 - Code Generation

## Angle Transformations

Several angle transformation functions have been changed to give users more flexibility when transforming angles and radiation patterns between azimuth-elevation coordinates and phi-theta coordinates.

- The `azel2phitheta` function converts directions from azimuth-elevation coordinates to phi-theta coordinates. `phitheta2azel` converts directions from phi-theta coordinates to azimuth-elevation coordinates. Updates to these functions let you use the `RotAx` input argument to select one of two definitions of phi-theta coordinates.

    - If `RotAx` is `true`, the phi angle is defined from the *y*-axis to the *z*-axis and the theta angle is defined from the *x*-axis toward the *yz*-plane.
    - If `RotAx` is `false`, the phi angle is defined from the *x*-axis to the *y*-axis and the theta angle is defined from the *z*-axis toward the *xy*- plane.
- The `azel2phithetapat` and `phitheta2azelpat` functions transform antenna patterns between azimuth-elevation and phi-theta coordinates. You can specify whether the antenna boresight is pointed along the *x*-axis or along the *z*-axis.

# R2019b

## Bicyclist Radar Reflection Model

This release introduces a bicyclist radar reflection model, `backscatterBicyclist`, to simulate backscattered radar signals from a bicycle and rider. The `backscatterBicyclist` model consists of moving point scatterers located on the wheels, pedals, the bicycle frame, as well as the legs and upper body components of the rider. Multiple reflections often occur when using high-resolution radars in automotive applications. Reflections from the model can be used to track bicyclists and can also be used to distinguish bicyclists from other types of objects such as motor vehicles and pedestrians. Modeling a bicyclist is important for automotive safety reasons and a good bicycle target model can yield high-fidelity detections. This release also introduces the corresponding Backscatter Bicyclist Simulink block.

## Detection Clustering

The `clusterDBSCAN` object clusters points in feature space using the Density-based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The clustering has applications in radar, sonar, and other areas. You can use clustering in radar to perform merging of multiple detections from extended targets which can reduce the computational burden on trackers and detection classifiers. This release also adds the corresponding Simulink block, DBSCAN Clusterer.

## Hybrid Beamforming

Two new functions, `omphybweights` and `ompdecomp`, support hybrid beamforming. The `omphybweights` function computes optimum weights for Multiple-Input and Multiple-Output (MIMO) beamforming based on an orthogonal matching pursuit (OMP) algorithm. The `ompdecomp` function decomposes a channel matrix using OMP.

## Sensor Array Analyzer

The interface for the **Sensor Array Analyzer** app has been redesigned with added capabilities. The new interface lets you display more information using multiple tabbed windows. The analyzer supports isotropic sonar projectors and hydrophones as array elements increasing its utility for sonar applications. The user can generate MATLAB code, export the array to a MATLAB workspace or a file, or import any valid array and element object or app session. The app displays several additional array characteristics: half-power beamwidth, first-null beamwidth, and sidelobe levels.

## Terrain Integrated Rough Earth Model Path Loss

This release adds the capability for computing radio frequency propagation loss over irregular terrain using the Terrain Integrated Rough Earth Model (TIREM) path loss function `tirempl`.

## Application Example: Deep Learning

This release contains a new feature application:

- The Pedestrian and Bicyclist Classification Using Deep Learning example shows how to classify pedestrians and bicyclists based on their micro-Doppler characteristics using deep learning networks and time-frequency analysis.

## Rename phased.BackscatterPedestrian Object

Starting with this release, the `phased.BackscatterPedestrian` object is being renamed to `backscatterPedestrian`. This is a name change only. The functionality remains the same and the original name will continue to work.

## Compatibility Considerations

The name `phased.BackscatterPedestrian` may not be supported in future releases.

# R2019a

**Version: 4.1**

**New Features**

**Bug Fixes**

## Pedestrian Radar Reflection Model

This release introduces a pedestrian radar reflection model, `phased.BackscatterPedestrian`, that simulate radar scattering from multiple points on a human body. Multiple reflections often occur when using high-resolution radars in automotive applications. These types of targets are called extended or multiple scatterer targets. Modeling a pedestrian is an important application for automotive safety reasons. Many users require a good pedestrian target model to obtain high fidelity detections. The `phased.BackscatterPedestrian` model consists of multiple scattering centers on the human body. Reflections from the model can be used to track pedestrians and can also be used for classification to distinguish pedestrians from other targets such as vehicles.

## Configure Antenna Element and Radiation Pattern Orientation

- This release provides more flexibility in specifying the orientation of antenna element patterns. You can rotate any antenna or microphone pattern using the `rotpat` function. For example, you can generate rotated patterns from patterns created by phased array element objects. Then, you can insert these patterns into the `phased.CustomAntennaElement` antenna element. You can also use the function to rotate radar cross-section patterns of targets.
- The `phased.CrossedDipoleAntennaElement` object creates an antenna whose orthogonal dipoles lie in the *yz*-plane of the antenna local coordinate system. Using the `RotationAngle` property, you can rotate the dipoles around the *x*-axis.
- For the `phased.ShortDipoleAntennaElement`, you can set the direction of the dipole using the `AxisDirection` and `CustomAxisDirection` properties.
- When a `phased.CustomAntennaElement` is used in an antenna array, setting the `MatchArrayNormal` property to true rotates the element normals into alignment with the array normal.

## Polarization Modes for Crossed-Dipole Antenna

In previous releases, the `phased.CrossedDipoleAntennaElement` object created only left circular polarized fields. Starting in this release you can also create right circular polarized or linear polarized fields using the `Polarization` property.

## Visualize Data Using Range, Doppler, and Angle Scopes

This release adds new data scopes to help you visualize processing results. In previous releases, visualizing range-Doppler, range-angle, or angle-Doppler response maps required the creation of `phased.RangeDopplerResponse`, `phased.RangeAngleResponse`, or `phased.AngleDopplerResponse` objects to generate the maps. You could then call the `plotResponse` object function to display the maps. This release provides `phased.RangeDopplerScope`, `phased.RangeAngleScope`, and `phased.AngleDopplerScope` to simplify this task, combining the creation and display in one operation.

In addition, this release introduces a range intensity scope, `phased.RTIScope`, and a Doppler intensity scope, `phased.DTIScope`. These scopes present a rolling display of intensity versus time. These objects extend the capabilities of the existing `phased.IntensityScope` scope.

## Create Waveform Library with Radar Waveform Analyzer App

The interface for the **Radar Waveform Analyzer** app has been redesigned with additional capabilities. Now you can create sets of waveform library specifications to use in the

`phased.PulseWaveformLibrary` and `phased.PulseCompressionLibrary` objects as well as the Pulse Waveform Library and Pulse Compression Library blocks. In addition, the `phased.PulseWaveformLibrary` object lets you export waveform specification data to use in the app.

## Radar Cross-Section Functions for Simple Shapes

In this release, several new functions — `rcssphere`, `rcsdisc`, `rcscylinder`, and `rcstruncone` — return the radar cross-sections for simple shapes — spheres, circular plates, cylinders, and truncated cones, respectively.

## Antenna Pattern Using Azimuth and Elevation Cuts

You can create antenna patterns from azimuth and elevation cuts to use in the `phased.CustomAntennaElement` rather than specifying a fully sampled 2-D pattern. To do this, use the `azelcut2pat` function. Specify the pattern along a slice at 0° azimuth and a slice at 0° elevation. The function interpolates for other directions.

## Application Examples: Radar coverage, beamforming, classification, SAR, and micro-Doppler effects

This release contains several new feature applications:

- The Planning Radar Network Coverage over Terrain example shows how to plan a radar network taking into account terrain information.
- The Introduction to Micro-Doppler Effects example is extended to include a section on using micro-Doppler to identify a moving pedestrian in an automotive radar scenario.
- The Stripmap Synthetic Aperture Radar (SAR) Image Formation example illustrates how stripmap synthetic aperture radar is used to improve resolution.
- The Processing Radar Reflections Acquired with the Demorad Radar Sensor Platform example shows how to process received FMCW echoes using the Analog Devices® DEMORAD Radar Sensor Platform.
- The Radar Target Classification Using Machine Learning and Deep Learning example shows how to use machine learning and deep learning for target classification.
- The 802.11ad Single Carrier Link with RF Beamforming in Simulink example simulates an 802.11ad single carrier link using beamformed phased array antennas.
- The Radar Waveform Classification Using Deep Learning example shows how to classify modulation types of generated synthetic waveforms using features from the Wigner-Ville distribution and a deep convolution neural network.

## Generate Single-Precision C Code for STAP Algorithms

This release supports single-precision processing when using System objects and functions in the Space-Time Adaptive Processing library. These are the `phased.STAPSMIBeamformer`, `phased.ADPCACanceller`, `phased.DPCACanceller`, and `phased.AngleDopplerResponse` System objects and the `dopsteeringvec` function.

# R2018b

**Version: 4.0**

**New Features**

**Bug Fixes**

## Pulse Compression Library: Switch pulse compression algorithms on PRI basis

The `phased.PulseCompressionLibrary` System object performs on-the-fly waveform range processing by performing matched filtering and stretch processing. The object enables you to switch compression algorithms on a PRI basis. This object can process waveforms generated by the `phased.PulseWaveformLibrary` System object introduced in the previous release.

This release also provides the corresponding Simulink block, Pulse Compression Library.

## Monopulse Feed: Combine channels and estimate target direction

This release adds two monopulse tracking System objects, `phased.MonopulseFeed` and `phased.MonopulseEstimator`, designed for direction finding using arbitrary arrays. These new System objects complement the functionality of the existing monopulse System objects, `phased.SumDifferenceMonopulseTracker` for uniform linear arrays and `phased.SumDifferenceMonopulseTracker2D` for uniform rectangular arrays. The `phased.MonopulseFeed` object forms sum and difference channels for monopulse processing of received signals on an arbitrary array. The `phased.MonopulseEstimator` object estimates the direction of arrival of narrowband signals from the sum and difference channels. By computing the ratio of the sum channel to the difference channels, monopulse processing can estimate the direction of a target with a resolution greater than the array beamwidth. Monopulse processing can also track targets by steering the array to follow a target.

The toolbox also includes the corresponding Monopulse Feed and Monopulse Estimator Simulink blocks.

## Alpha-Beta Filter: Predict and track target location and speed

This release introduces an alpha-beta tracking filter object, `phased.AlphaBetaFilter`. The alpha-beta filter is a fixed-gain tracking filter and is a simplified form of the linear Kalman filter but requires less computation. The filter works with constant-velocity or constant-acceleration motion models.

## Code Generation: Generate single precision C code for receive signal processing algorithms

This release supports single-precision processing by the System objects and blocks in the Detection, Beamforming, and Direction of Arrival Estimation libraries.

## Range-Angle Response: Generate and plot range-angle maps

This release adds a `phased.RangeAngleResponse` System object and corresponding Range Angle Response Simulink block. You can use this object to create a range-angle data map that adds to the capabilities of the existing `phased.RangeDopplerResponse` and `phased.AngleDopplerResponse` maps. The object also lets you visualize range-angle responses using the `plotResponse` object function.

## Scattering MIMO Channel: Model polarization effects with scattering matrix

The `phased.ScatteringMIMOChannel` System object can now simulate the propagation and scattering of polarized signals in a MIMO environment. Polarized signals are commonly employed in the 5G wireless and radar communities.

## Platform Trajectories: Build waypoint trajectories and scanning patterns

This release enhances some of the capabilities of the `phased.Platform` System object and Motion Platform Simulink block. The new capabilities let you model mechanical scanning radars. In addition, you can also specify platform trajectories using waypoints as well as motion models.

## Range Response Grid Centering: Choose origin of range grid

This release adds range centering to the `phased.RangeResponse` and `phased.RangeDopplerResponse` System objects and their corresponding blocks, Range Response and Range Doppler Response. The new `phased.RangeAngleResponse` System object and Range Angle Response block also include this capability.

## Application Examples: Multifunction radar, micro-Doppler effects, and polarimetric Doppler weather radar

This release contains several new feature applications:

- The Search and Track Scheduling for Multifunction Phased Array Radar example shows how to design a radar system to perform both searching and tracking.
- The Introduction to Micro-Doppler Effects example makes use of the micro-Doppler effect caused by target rotation present in the radar returns of targets.
- The Simulating a Polarimetric Radar Return for Weather Observation example shows how to simulate the return from a polarimetric Doppler weather radar.

## Algorithm Acceleration: Use Dataflow to speed up simulations

Two new examples illustrate how you can use Dataflow to increase the speed of simulations:

- The Multicore Simulation of Audio Beamforming System example shows how to beamform signals received by an array of microphones to extract a desired speech signal in a noisy environment. The example uses dataflow to take advantage of inherent parallelism in the system and to run a multi-threaded audio processing system simulation.
- The Multicore Simulation of Monostatic Radar System example shows how to use dataflow to partition a monostatic radar model to run a multi-threaded simulation. The example simulates a simple monostatic radar with one target.

# R2018a

**Version: 3.6**

**New Features**

**Bug Fixes**

## Waveform Library: Switch transmit waveforms on PRI basis during simulation

This release introduces the `phased.PulseWaveformLibrary` System object and its corresponding Pulse Waveform Library Simulink block. You can create a library of different types of pulse waveforms during a simulation and then transmit any one of the waveforms after each pulse repetition interval (PRI). You can add rectangular, linear FM, stepped FM, phase-coded, and custom waveforms to the library. This waveform agility enables you to match waveforms to a changing radar or sonar scenario and is useful for simulating multi-function radars. For example, you can transmit a new waveform or you can modify the waveform pulse length, and pulse repetition frequency of the current waveform to change range resolution or Doppler limits. Another example is changing the transmit frequency to mitigate jammers and interference.

## PRF Agility: Dynamically select PRF during simulation for STAP and clutter

You can change the pulse repetition frequency (PRF) during a simulation when you use STAP, clutter, or stretch processing System objects and Simulink blocks.

The `PRFSource` property of the `phased.STAPSMIBeamformer`, `phased.ADPCACanceller`, `phased.DPCACanceller`, `phased.AngleDopplerResponse`, and `phased.StretchProcessor` System objects lets you set the PRF using the `PRF` property at creation time or using an input argument at execution time.

This capability is also supported by the SMI Beamformer, ADPCA Canceller, DPCA Canceller, Angle Doppler Response, and Stretch Processor Simulink blocks by using the **Specify PRF as** parameter.

You can output PRF values from any pulse waveform System object at execution time using an output argument. To enable this, use the `PRFOutputPort` property of the `phased.LinearFMWaveform`, `phased.PhaseCodedWaveform`, `phased.RectangularWaveform`, and `phased.SteppedFMWaveform` System objects. The corresponding feature for Simulink is enabled by selecting the **Enable PRF output** check box of the Linear FM Waveform, Phase Coded Waveform, Rectangular Waveform, and Stepped FM Waveform blocks.

In this release, you can select PRF input modes for the `phased.ConstantGammaClutter` System object by using the `PRFSelectionInputPort` property and for the Constant Gamma Clutter block by using the **Enable PRF selection input** parameter.

## Array Pattern Scaling: Apply field or directivity pattern to radiated or collected signals

The release introduces an alternative scaling of the received power in radar system simulations. At present, the received power is scaled by the field pattern. You can now scale the input power to an array by the directivity pattern in addition to the field pattern. This computation is useful when comparing a simulation result to the solution of the radar equation. You can use this feature by setting the `SensorGainMeasure` property of the `phased.Radiator`, `phased.WidebandRadiator`, `phased.Collector`, and `phased.WidebandCollector` System objects. For Simulink, set the **Sensor gain as** parameter in the Narrowband Transmit Array, Narrowband Receive Array, Wideband Transmit Array, and Wideband Receive Array blocks.

## Application Examples: Model radar warning receiver and target classification systems

This release introduces several new featured application examples that show how to analyze radar signals, classify targets, and mitigate interference in radar systems.

- The Signal Parameter Estimation in a Radar Warning Receiver example shows how to extract signal parameters in a simulation of a radar warning receiver.
- The Radar Target Classification Using Machine Learning example demonstrates how to employ machine learning and wavelets to classify targets.
- The Interference Mitigation Using Frequency Agility Techniques example shows how to mitigate interference in a radar system by employing frequency agility.

## PRF Agility: Improve staggered PRF support in Simulink

This release provides better timing support for Simulink simulations that implement pulse repetition frequency (PRF) agility. This capability also makes these types of simulations easier to construct and understand. Set the **Source of simulation sample time** parameter to `Derive from waveform parameters` to obtain the simulation elapsed time from the waveform instead of the Simulink engine. Then, the block runs at a variable rate determined by the PRF of the selected waveform and the elapsed time is variable. When you set the **Source of simulation sample time** parameter to **Inherit from Simulink engine**, the block runs at a constant elapsed time. This parameter appears when you select **Enable PRF selection input** in the Linear FM Waveform, Phase Coded Waveform, Rectangular Waveform, Stepped FM Waveform, and Constant Gamma Clutter blocks.

# R2017b

## Independent Subarray Steering: Model multifunction array aperture with subarrays

System objects that support subarrays, such as `phased.PartitionedArray` and `phased.ReplicatedSubarray`, now allow independent subarray steering. By applying weights to the individual elements in each subarray, you can steer subarrays in different directions. Several Simulink blocks also support subarray steering.

These System objects and object functions support independent subarray steering:

| |
|---|
| `phased.Radiator` and `phased.WidebandRadiator` |
| `phased.Collector` and `phased.WidebandCollector` |
| `phased.PartitionedArray`, `phased.ReplicatedSubarray`, and their `pattern`, `patternAzimuth`, `patternElevation` and `directivity` object functions |
| `phased.ArrayGain`, `phased.ArrayResponse`, and `phased.SteeringVector` |
| `phased.ConstantGammaClutter` and `phased.gpu.ConstantGammaClutter` |

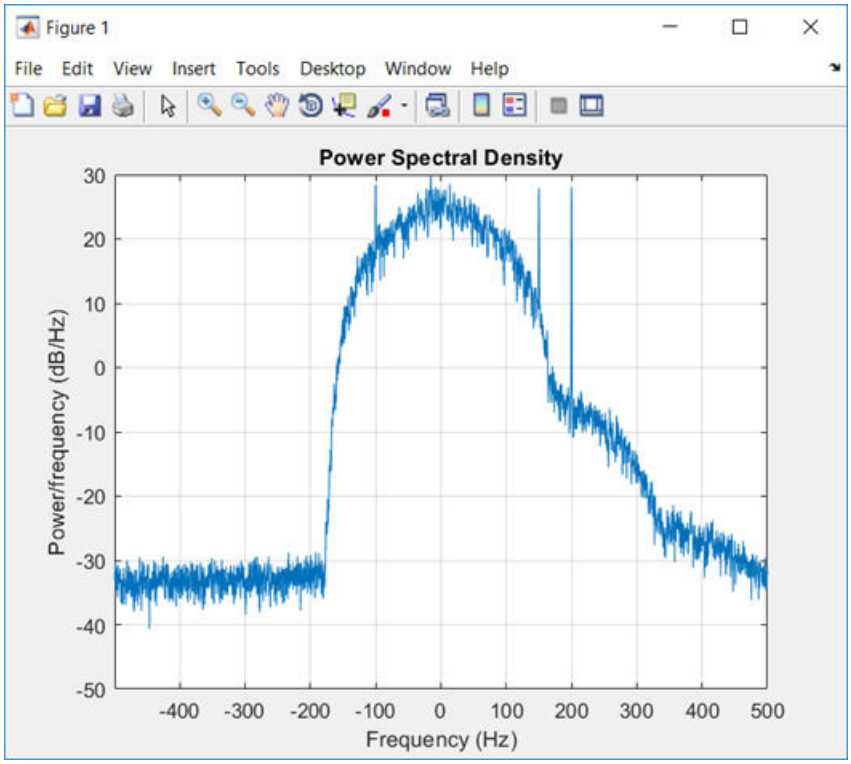
These Simulink blocks support independent subarray steering:

| |
|---|
| Narrowband Receive Array and Narrowband Transmit Array |
| Wideband Receive Array and Wideband Transmit Array |
| Constant Gamma Clutter and GPU Constant Gamma Clutter |

## Sonar Noise Source: Model noise radiated by surface and underwater sources

You can now create an acoustic source for generating underwater noise in a sonar simulation. The `phased.UnderwaterRadiatedNoise` System object generates broadband and discrete tonal noise. Tonal noise can model, for example, shaft rotation or machinery. Use this System object to model noise generated by both surface and submerged vessels.

This figure shows a sample generated spectrum containing broadband noise and tones.




## Dual Polarization: Model radar and wireless systems with polarization diversity

Use dual polarization to transmit and receive independent signals from an antenna or antenna array using orthogonal polarizations. This capability is useful for remote sensing and target recognition and classification. The feature is enabled in the narrowband and wideband radiator System objects, `phased.Radiator` and `phased.WidebandRadiator`, and the narrowband and wideband collector objects, `phased.Collector` and `phased.WidebandCollector` when you set the `Polarization` property to `'Dual'`. Use these System objects in conjunction with antenna elements that support polarized electromagnetic radiation: `phased.CustomAntennaElement`, `phased.ShortDipoleAntennaElement`, and `phased.CrossedDipoleAntennaElement`.

The `Polarization` property replaces the `EnablePolarization` property. This property lets you select nonpolarized, polarized, or dual-polarized output.

## Sonar Equation: Estimate maximum range, SNR, transmission loss, and source level of a sonar system using functions and app

This release introduces a sonar equation app and sonar equation functions.

The **Sonar Equation Calculator** app is a convenient way to predict sonar performance using the sonar equation. The sonar equation relates received signal-to-noise ratio (SNR), source level, noise level, transmission loss, receiver directivity, and target strength. The app computes SNR, transmission loss, range, or source level in terms of the other quantities. You can compute expected SNR for given sonar parameters, or you can invert the equation to compute a needed source level or maximum transmission loss to meet a particular SNR requirement. You can enter SNR directly or

compute SNR in terms of probability of detection and probability of false alarm. The app takes into account spherical and cylindrical spreading in a sound channel and sound absorption due to chemical processes in seawater.

The toolbox also provides five functions for using the sonar equation on the command line.

**Sonar Equation Functions**

| Name | Purpose |
|---|---|
| `sonareqsl` | Compute the required source level to achieve a desired SNR in terms of transmission loss, noise level, receiver directivity, and target strength. |
| `sonareqsnr` | Compute the received SNR from source level, transmission loss, noise level, receiver directivity, and target strength. |
| `sonareqtl` | Compute the transmission loss that produces a given SNR in terms of source level, noise level, receiver directivity, and target strength. |
| `range2tl` | Compute transmission loss from range, taking into account geometrical spreading and sound absorption. The spreading includes spherical and cylindrical spreading in a sound channel of finite depth. |
| `tl2range` | Compute range that produces a given transmission loss taking into account the geometrical spreading and sound absorption. |

## Application Examples: Model wireless, automotive and MIMO radar, and EW systems

This release introduces five new application examples that show how to model wireless communications, automotive radar, MIMO radar, and EW systems.

- The Introduction to Hybrid Beamforming example shows how to use hybrid beamforming to improve data throughput in a wireless communications system.
- The Antenna Array Beam Scanning Visualization on a Map example shows how to visualize the changing coverage map of an antenna array used in communications as it scans a sweep of angles. The antenna array is created using Antenna Toolbox™.
- The Radar Signal Simulation and Processing for Automated Driving example shows how to model the hardware, signal processing, and propagation environment of a radar used in a driving scenario. The example uses Automated Driving System Toolbox™.
- The Increasing Angular Resolution with MIMO Radars example shows how forming a virtual array can improve the angular resolution of real arrays.
- The Frequency Agility in Radar, Communications, and EW Systems example uses frequency agility to reduce the effects of interference in radar, communications, and EW systems.

## Heterogeneous sensor arrays support code generation

Arrays created using the `phased.HeterogeneousConformalArray`, `phased.HeterogeneousULA`, and `phased.HeterogeneousURA` System objects now support code generation. Heterogeneous

arrays let you construct phased arrays consisting of different sensor elements. All phased arrays now support code generation for transmitting and receiving nonpolarized electromagnetic or acoustic radiation.

# R2017a

**Version: 3.4**

**New Features**

**Compatibility Considerations**

## Scattering MIMO Channel: Model multipath signal propagation through spatially spread scatterers

The `phased.ScatteringMIMOChannel` System object models multipath signal propagation often encountered in wireless communications systems. This object simulates the propagation of a signal from a transmitter array to multiple scattering centers and back to a receiver array. Signals propagate along line-of-sight paths. Use this object to model loss due to rain, fog, and atmospheric gases. The toolbox also includes the corresponding Simulink block, Scattering MIMO Channel.

In addition, you can generate a MIMO channel matrix using the `scatteringchanmtx` function. The `diagbfweights` function returns precoding and combining weights for spatial multiplexing within a MIMO channel. Use these weights to diagonalize a MIMO channel matrix into orthogonal and independent sub-channels. This technique improves channel capacity. In addition, the `waterfill` function improves MIMO channel capacity by optimally distributing power into multiple channels.

## Sonar Systems: Model hydrophones, projectors, underwater propagation, and targets

You can now simulate sonar systems using Phased Array System Toolbox. For example, you can simulate an end-to-end monostatic active sonar or a passive surveillance sonar system:

- `phased.IsotropicProjector` models an isotropic sonar projector. Use this System object to create a transmitting phased array using any of the array System object types, such as `phased.ULA`.
- `phased.IsotropicHydrophone` models an isotropic sonar hydrophone. You can also use this System object as part of a receiving phased array.
- `phased.IsoSpeedUnderwaterPaths` and `phased.MultipathChannel` model multipath propagation in an underwater channel.
- `phased.BackscatterSonarTarget` models backscattered signals from an underwater target. The object supports multiple angle-dependent target strength matrices. Backscattering applies to monostatic sonar applications. For performance improvement, you can simultaneously compute scattering for all paths in a multipath environment.

## Range and Doppler Estimation: Measure target range and speed

The `phased.RangeResponse` System object models matched-filter and CW responses to time-sampled data. This object computes range-only responses, in contrast to the `phased.RangeDopplerResponse` object, which provides range and Doppler processing. Range-only responses are useful when your system simulates noncoherent pulse transmission for which Doppler estimation cannot be computed. The `phased.RangeEstimator` and `phased.DopplerEstimator` provide high-resolution target range and Doppler estimation for detected targets.

The toolbox also includes corresponding Simulink blocks: Range Response, Range Estimator, and Doppler Estimator.

The utility function `bw2range` converts signal bandwidth to range resolution.

Modifications to the `phased.CFARDetector` and `phased.CFARDetector2D` System objects support the range and Doppler estimators by outputting detection localization information and noise power estimates for the detection.

An option added to the `phased.RangeDopplerResponse` object lets you specify pulse-repetition frequencies (PRF). Previously, PRF was computed automatically from the sample rate and pulse duration. This option enables the processing of time-gated input data.

## Custom antenna element phase pattern

The `phased.CustomAntennaElement` System object now supports phase angles for the custom antenna elements, `phased.CustomAntennaElement`. Use the `PhasePattern` and `MagnitudePattern` properties to specify the phase and magnitude response of the antenna to a nonpolarized signal. The `MagnitudePattern` property replaces the `RadiationPattern` property, with no change in behavior.

# R2016b

**Version: 3.3**

**New Features**

**Bug Fixes**

## Wideband Targets: Model frequency-dependent and angle-dependent RCS

The `phased.WidebandBackscatterRadarTarget` System object models the reflection of wideband signals from targets in monostatic radar simulations. This System object extends to wideband signals the capabilities of the narrowband `phased.BackscatterRadarTarget` System object.

You can specify angle-dependent radar cross-section *(RCS)* models for nonpolarized fields or scattering pattern models for polarized fields. For nonpolarized signals, you specify the RCS pattern as a MATLAB array dependent on incident angles and frequency. For polarized signals, you specify a set of three arrays describing the *HH*, *VV*, and *HV* scattering patterns. For the corresponding Simulink block, you can use Wideband Backscatter Radar Target.

## Multipath Propagation: Model atmospheric effects on narrowband and wideband signals in two-ray channels

The `phased.WidebandTwoRayChannel` System object models wideband signal propagation when the signal channel includes a planar boundary, such as a flat earth. The wideband two-ray channel propagation model includes atmospheric attenuation caused by fog, rain, or atmospheric gases. This release also adds atmospheric attenuation models to the narrowband signal `phased.TwoRayChannel` System object. For the corresponding Simulink block, see Wideband Two-Ray Channel.

## 2-D CFAR Detector: Perform constant false alarm rate detection on range-Doppler maps

The `phased.CFARDetector2D` System object performs constant false alarm rate detection on two-dimensional data. Like the `phased.CFARDetector` System object, the 2-D CFAR object calculates the detection threshold required to produce a constant false alarm rate. The CFAR threshold is computed by averaging over signal-free training cells in a two-dimensional region surrounding the test cell. For the corresponding Simulink block, use 2-D CFAR Detector.

## MUSIC DOA Estimation: Resolve closely-spaced narrowband sources

This release adds the high-resolution direction-finding algorithm, MUSIC, for resolving closely-spaced narrowband sources. The `phased.MUSICEstimator` System object performs MUSIC estimation on uniform linear arrays. For two-dimensional arrays, use `phased.MUSICEstimator2D`.

For the corresponding Simulink blocks, see ULA MUSIC Spectrum for uniform linear arrays and MUSIC Spectrum for two-dimensional arrays.

You can also use the `musicdoa` function to compute directions of arrival of signals on ULA arrays based on the sensor covariance matrix.

## Low Sidelobe Taper: Taylor tapers for circular planar arrays

Use the `taylortaperc` function to create Taylor tapers, which you can then apply to circular planar apertures. Arrays with Taylor tapers maintain nearly constant sidelobe levels.

## Periodic and Cross-ambiguity Functions: Compute ambiguity functions for periodic signals

The `pambgfun` function computes the *p*-period ambiguity function for periodic signals. In addition, the enhanced ambiguity function, `ambgfun`, computes the crossambiguity between two signals.

## Generalized Sidelobe Canceler: Adaptively suppress interference

The generalized sidelobe canceler (GSC) System object, `phased.GSCBeamformer`, adaptively suppresses directional interference. GSC is an optimized version of the adaptive linear-constrained minimum variance (LCMV) beamformer, whereby the constrained optimization problem is replaced by an unconstrained one. For the corresponding Simulink block, see GSC Beamformer.

## PRF Agility: Dynamically select PRF during simulation

You can run simulations in *pulse mode* with a dynamically changing pulse repetition frequency *(PRF)*. This feature lets you simulate, for example, an MTI radar with staggered PRF waveforms. Pulse mode is available when you use one of the pulsed waveform System objects: `phased.LinearFMWaveform`, `phased.PhaseCodedWaveform`, `phased.RectangularWaveform`, or `phased.SteppedFMWaveform`. To run in pulse mode, specify the waveform `OutputFormat` property as `Pulses`. Then, you can specify the `PRF` property as a row vector of varying values. When running a simulation in pulse mode the number of samples per pulse will change when you specify a variable PRF. In contrast, when you specify the `OutputFormat` property as `Samples`, the number of samples is set by the `NumSamples` property and remains fixed. PRF agility also applies to the corresponding Simulink blocks.

## System Objects: Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object, you can call the object with arguments, as if it were a function. The `step` method will continue to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `phased.WidebandCollector` System object named `collector`, then you call the System object as a function with that name

```
collector = phased.WidebandCollector('Sensor',array,'SampleRate',fs);
x = collector(y,incident_ang);
```

The equivalent operation using the `step` method is:

```
collector = phased.WidebandCollector('Sensor',array,'SampleRate',fs);
x = step(collector,y,incident_ang);
```

When the `step` method has the System object as its only argument, the function equivalent will have no arguments. This function must be called with empty parentheses. For example,

```
waveform = phased.LinearFMWaveform('PulseWidth',10.0e-6);
wv = step(waveform);
```

and

```
waveform = phased.LinearFMWaveform('PulseWidth',10.0e-6);
wv = waveform();
```

perform equivalent operations.

# R2016a

**Version: 3.2**

**New Features**

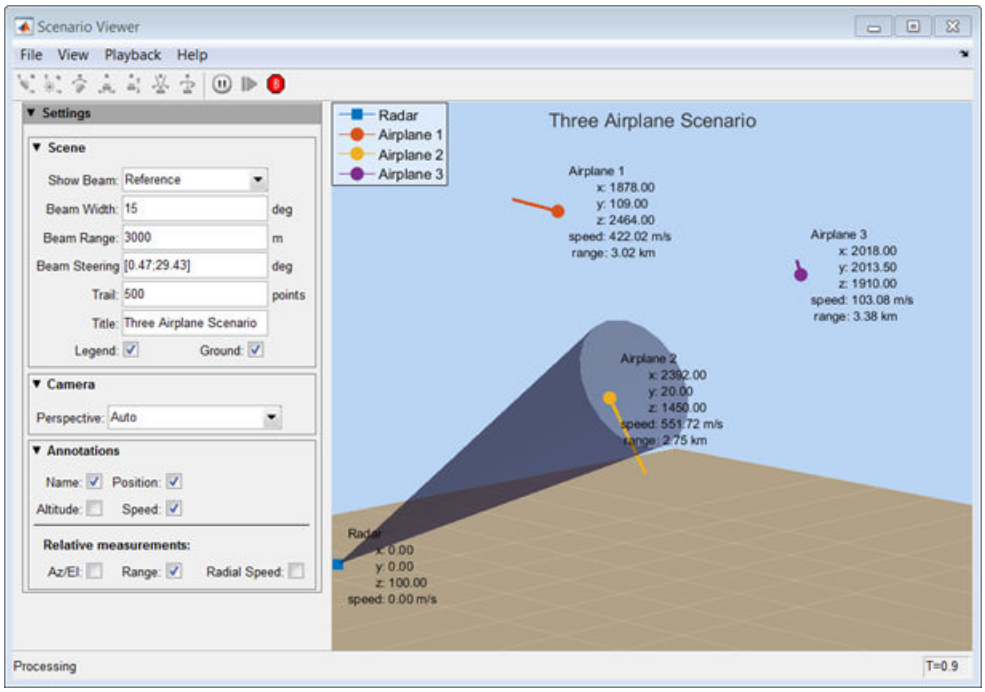
**Bug Fixes**

## Scenario Viewer: Visualize radar and target trajectories

The `phased.ScenarioViewer` System object is a UI tool for visualizing the trajectories of multiple objects, receivers, and transmitters in radar scenarios. The System object creates a 3-D display showing the motion of radars and targets over time. Use this display to:

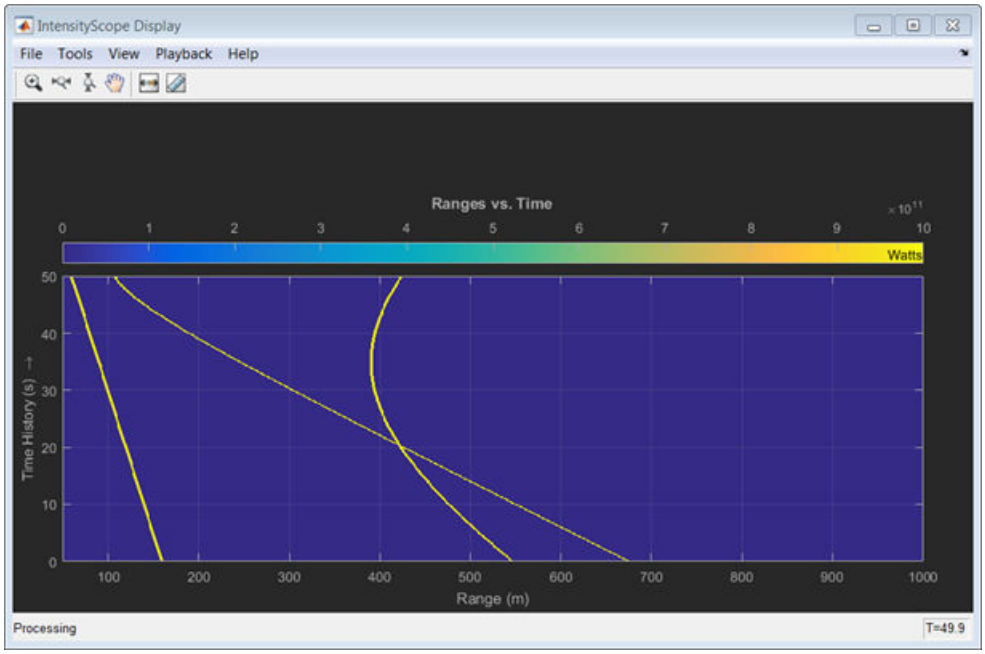
- annotate scenario objects with indicators such as position, speed, range, and angle
- toggle the display of radar beams and dynamically alter their pointing directions
- label the scenario objects with custom tags and motion history tracks
- change all parameters during the simulation using the UI panel

Although designed for radar scenarios, you can also use the UI to visualize sonar and microphone models. You can change all parameters during the simulation by using the UI panel.




## Intensity Scope: Visualize range-time-intensity (RTI) and Doppler-time-intensity (DTI) images

The `phased.IntensityScope` System object lets you create RTI and DTI displays for radar and sonar. An RTI display shows echo intensity as a function of time and range. A DTI display shows echo intensity as a function of time and Doppler shift, or speed. You can also visualize angle-time-intensity data or display spectrograms.




## Atmospheric Loss: Model rain, fog, and atmospheric gas attenuation of RF signals

The `phased.LOSChannel` System object lets you simulate line-of-sight (LOS) propagation of narrowband RF signals. The model includes attenuation due to dry air, water vapor, rain, fog, and geometric spreading. The `phased.WidebandLOSChannel` System object performs the same function for wideband RF signals. These System objects correspond to the `phased.FreeSpace` and `phased.WidebandFreeSpace` System objects, which model propagation in vacuum. In addition, the toolbox provides two associated Simulink blocks: LOS Channel and Wideband LOS Channel. When you want to perform analytic modeling, use the `gaspl` function to simulate attenuation due to atmospheric gases. The `fogpl` and `rainpl` functions let you model attenuation due to fog and rain, respectively.

## Backscattered Radar Target: Model backscattered radiation with angle-dependent, custom RCS patterns

Using the `phased.BackscatterRadarTarget` System object, you can model an angle-dependent radar cross-section (RCS) or scattering pattern model for monostatic radar simulations. For nonpolarized signals, specify the RCS pattern as a single *M*-by-*N* matrix. For polarized signals, you specify a set of three matrices describing *HH*, *VV*, and *HV* scattering patterns. The entries in the matrices specify the RCS value for each azimuth and elevation angle with respect to the local target coordinate system. The toolbox provides an associated Simulink block, Backscatter Radar Target.

## Phase Shift Quantization: Model effects of quantized beamformer weights on array patterns and responses

In this release, you can model beamformer and steering vector weights to finite precision. This feature appears in the System objects listed in the table and in the corresponding Simulink blocks.

| System Objects with Phase Quantization Property | |
|---|---|
| phased.ADPCACanceller | phased.PartitionedArray |
| phased.BeamscanEstimator | phased.PhaseShiftBeamformer |
| phased.BeamscanEstimator2D | phased.ReplicatedSubarray |
| phased.DPCACanceller | phased.STAPSMIBeamformer |
| phased.MVDREstimator | phased.SteeringVector |
| phased.MVDREstimator2D | phased.SumDifferenceMonopulseTracker |
| phased.MVDRBeamformer | phased.SumDifferenceMonopulseTracker2D |

In addition, the steering vector function, `steervec`, the conventional beamformer weights function, `cbfweights`, and the minimum-variance distortionless response weights function, `mvdrweights`, include a quantization bit argument.

In analytic models which use ideal components, the phase shifting required for precise array steering and beamforming is computed to infinite precision. In practical applications that model real radar components, the element phase shifting is computed to finite precision. Finite precision gives rise to quantization effects that can affect the beamforming and steering performance of an array. In some simulations, it is important to model quantization effects. Quantized phase shifters are categorized by the number of bits in the phase component of the weight vector.

## Array Orientations: Set array pointing direction

Two features in this release give you more options for positioning elements on uniform arrays:

- The `ArrayAxis` property lets you specify the position of ULA elements along the *x*-axis, *y*-axis, or *z*-axis of the local coordinate system. Previously, elements were located only along the *y* axis. This property applies to the `phased.ULA` and `phased.HeterogeneousULA` System objects.
- The `ArrayNormal` property lets you place the elements of uniform planar arrays on the *x*-*y*-plane, *y*-*z*-plane, or the *z*-*x*-plane. Previously, elements were located only on the *z*-*x* plane. In these cases, the array normal axis points along the *z*, *x*, or *y* axes, respectively. This property applies to the `phased.URA`, `phased.UCA` and `phased.HeterogeneousURA` System objects.
- These features appear in the Simulink blocks that use uniform arrays.

To obtain the direction normal vector for the elements of these arrays, use the `getElementNormal` method belonging to each System object.

## Dynamic PRF: Change pulse repetition frequencies

You can change the pulse repetition frequency (PRF) at each call to the `step` method for these pulse waveform System objects: `phased.RectangularWaveform`, `phased.LinearFMWaveform`, `PhaseCodedWaveform`, and `phased.SteppedFMWaveform`. When initializing the waveforms, you specify a vector of PRFs. In the `step` method call, you then specify an index into this vector.

In addition, the `DutyCycle` property lets you specify pulse duration in terms of duty cycle. Previously, you specified pulse duration in the `PulseWidth` property. When you specify `DutyCycle` and `PRF` properties, the System object computes the pulse duration for you. The `DutyCycle` property applies to the `phased.RectangularWaveform`, `phased.LinearFMWaveform`, and `phased.SteppedFMWaveform` waveforms.

## Short Dipole Orientation: Align dipole direction along any axis

You can align the dipole direction of the `phased.ShortDipoleAntennaElement` System object along the *x*-axis, *y*-axis, and *z*-axis.

# R2015b

## Wideband Radiator: Radiate wideband signals from arrays and elements

The `phased.WidebandRadiator` System object lets you create wideband signals that radiate from an element or array source into different spatial directions. This System object is the wideband counterpart of the narrowband `phased.Radiator` System object. The radiator employs subband frequency processing to generate wideband propagating signals. Subband processing works in the frequency domain by dividing the signal spectrum into frequency bands and processing each band as if it were a narrowband signal.

This release also includes the corresponding Wideband Transmit Array Simulink block.

The `phased.WidebandRadiator` System object is one of several System objects introduced in this release to process wideband signals.

## Wideband Free Space: Propagate wideband signals through free space

The `phased.WidebandFreeSpace` System object propagates wideband signals in free space along line-of-sight paths. The System object models time-delay, Doppler, and propagation loss and is the wideband counterpart of the narrowband `phased.FreeSpace` System object. The wideband free-space propagator uses subband frequency processing to propagate signals. Subband processing works in the frequency domain by dividing the signal spectrum into frequency bands and processing each band as if it were a narrowband signal.

This release also includes the corresponding Wideband Free Space Simulink block.

## Wideband MVDR Beamformer: Perform frequency-domain MVDR beamforming on wideband signals

The `phased.SubbandMVDRBeamformer` System object performs minimum-variance distortionless response beamforming on wideband signals. This System object is the wideband counterpart of the narrowband `phased.MVDRBeamformer` System object. The wideband MVDR System object uses subband frequency processing to beamform a signal. Subband processing works in the frequency domain by dividing the signal spectrum into frequency bands and processing each band as if it were a narrowband signal.

This release also includes the corresponding Subband MVDR Beamformer Simulink block.

## Wideband DOA Estimator: Estimate the arrival angle of wideband signals using generalized cross-correlation

The `phased.GCCEstimator` System object estimates the direction of arrival (DOA) of wideband signals on an array. The System object uses the generalized cross-correlation with phase transform (*GCC-PHAT*) algorithm to determine the time delay of coherent signals from a common source arriving between pairs of sensors in an array. The algorithm estimates time delays from the cross-correlation peaks between signals. The direction of arrival follows directly from the computed time delays. The *GCC-PHAT* algorithm improves the sharpness of the correlation peak over ordinary correlation by whitening the signal spectrum.

In addition, the new `gccphat` function computes the time delay between two coherent signals from a common source arriving at different sensors.

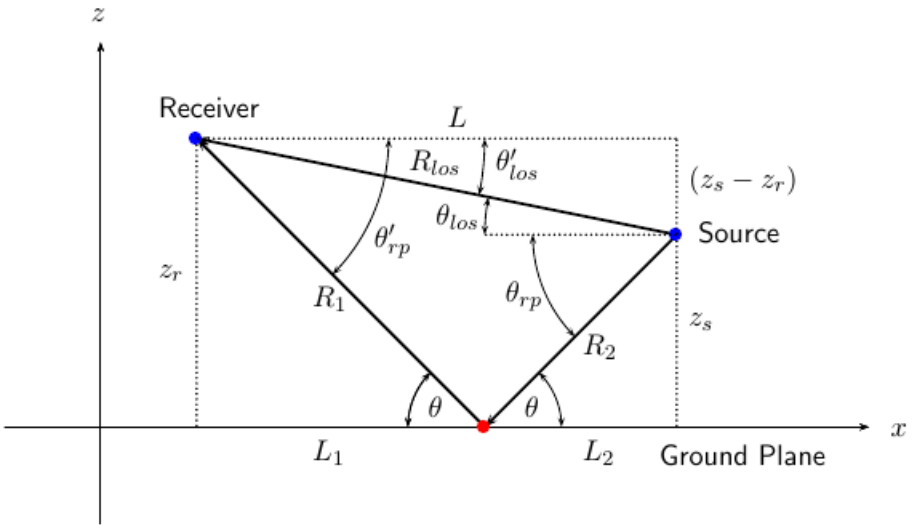
This release includes the corresponding GCC DOA and TOA Simulink block.

## Two-Ray Channel: Propagate signals along line-of-sight and ground-reflected paths

The `phased.TwoRayChannel` System object propagates signals in a channel with a single reflecting boundary. This type of boundary often occurs with radar signals reflecting from a ground plane or with acoustic signals reflecting from the ocean surface. This System object models the line-of-sight propagation path and the reflected propagation path. The model assumes flat-earth geometry and works with scalar field or polarized electromagnetic fields. To compute reflection loss for scalar fields, you specify a ground reflection loss parameter. For polarized electromagnetic fields, specify the relative dielectric permittivity.

In addition, the `rangeangle` function now computes the path distance and arrival directions for the line-of-sight and reflected ray paths. The `phased.TwoRayChannel` System object and `rangeangle` function work in air or underwater environments provided the signals are non-polarized fields and the propagation speed is appropriate for the medium.

All propagation paths follow straight lines between boundaries. This illustration illustrates line-of-sight (los) and reflected paths (rp) and arrival and emittance angles.




This release also includes the corresponding Two-Ray Channel Simulink block.

## Improved Accuracy of Free Space Propagation: Implement fractional sample-time delay

This release changes how the `phased.FreeSpace` System object models propagation delays. In previous versions, propagation delays were always an integral number of sample times. This version outputs signals having fractional sample time delays. This change leads to a more accurate estimate of the actual propagation time for a signal.

## Compatibility Considerations

The propagated signal output can differ from the output of previous versions.

## Platform Motion: Model acceleration of targets and signal sources

The `phased.Platform` System object now models platform acceleration in addition to the existing constant-velocity model. Acceleration is also included in the corresponding Motion Platform Simulink block.

## Wideband Collector: Subband selection changes

In the `phased.WidebandCollector` System object, when performing subband processing, you can choose the number of frequency subbands. Alternatively, the number of subbands can be automatically calculated. Similarly, you can choose the number of subbands in the corresponding Wideband Receive Array Simulink block.

## Compatibility Considerations

The method for computing the default value for the number of frequency subbands has changed, potentially leading to different results. You can minimize the differences by setting the `NumSubbands` property value equal to the signal length.

## Antenna Toolbox Integration: Changes to field values

Field values that you generate when using Antenna Toolbox antennas in a Phased Array System Toolbox array have changed. Previous array computations used the actual field value generated by the antenna. The new computation uses a normalized field value created by dividing the field by its maximum value over all spatial directions. Spatial directions are sampled over a 5° uniform grid in azimuth and elevation.

## Compatibility Considerations

Computed values of radiated or received fields are changed. Antenna and array directivity pattern values are not change because they are already normalized quantities.

## Platform Motion: Changes to orientation dynamics

In previous versions of the `phased.Platform` System object, the orientation matrix returned by the `step` method remained incorrectly fixed to the value of the `OrientationAxes` property specified during System object construction. (The `step` method returns the platform orientation matrix when

the `OrientationAxesOutputPort` property is set to `true`.) In this release, the returned orientation matrix is properly updated at every call to the `step` method. The orientation axes are rotated if the platform undergoes curvilinear motion. Finally, the name of the `OrientationAxes` property is changed to `InitialOrientationAxes` to emphasis that the property specifies the initial value of the orientation matrix.

## Compatibility Considerations

You should replace `OrientationAxes` by `InitialOrientationAxes`. In order to maintain the original behavior, set the `OrientationAxesOutputPort` property to `false` and always use the initial matrix that you specified in `InitialOrientationAxes` for all downstream processing.

# R2015a

**Version: 3.0**

**New Features**

**Bug Fixes**

## Antenna Toolbox Integration: Use elements from Antenna Toolbox to design antenna arrays

This release lets you use antenna elements from Antenna Toolbox in your phased array models and simulations. For example, to create a four-element array of helix antennas, enter

```
sAnt = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
sArray = phased.ULA('Element',sAnt,'NumElements',4);
```

just as you would use antennas from Phased Array System Toolbox. Use of Antenna Toolbox requires an additional license. For more information on using this toolbox, go to Antenna Toolbox.

## Array Calibration: Calibrate array element gain, phase, and position using pilot sources

This release introduces a function to perform array calibration. You can use the `pilotcalib` function to calibrate individual element gains, phases, and positions. This algorithm implements the pilot source technique and is commonly used for array calibration. The algorithm works by setting up multiple transmitters in known directions and using the array to receive the signals from those transmitters. Because these transmitters are in known directions, you can compute the expected received signal for the array and compare it to the actual received signal. From these differences, you can calibrate the array.

## Uniform Circular Array: Perform beamforming and root-MUSIC direction-of-arrival estimation

This release introduces the `phased.UCA` System object to simulate uniform circular arrays (UCA). A uniform circular array has its elements equally spaced around a circle. The element normals point along the outward radius of the array. You can specify the number of elements and the radius of the array. From these two quantities, the object computes the position of the elements. This array accepts any polarized or nonpolarized element types and accepts complex-valued element tapers. You can use a UCA System object as an array for beamforming and direction-of-arrival in System objects such as `phased.PhaseShiftBeamformer` and `phased.LCMVBeamformer`.

In addition, enhancements to the `phased.RootMUSICEstimator` System object let you employ root-MUSIC direction finding algorithms on uniform circular arrays.

## Model Simplification: Simulate multiple targets, platforms, and propagation channels using single blocks

Several blocks and System objects have been vectorized to support multitarget scenarios. When simulating multiple platforms and targets, you can combine separate blocks describing target cross-sections, platform motion, or channel propagation into single blocks. This change can simplify complex models. In addition, you can parameterize the number of targets to enable easy modification.

The following blocks and System objects now allow vectorized inputs and outputs:

- Radar Target
- Motion Platform
- Free Space

- `phased.RadarTarget`
- `phased.Platform`
- `phased.FreeSpace`

## Continuous MFSK Waveform: Estimate range and speed simultaneously for multiple targets

This release introduces a Multiple Frequency Shift Keying (MFSK) waveform, `phased.MFSKWaveform` System object and the corresponding Simulink block, MFSK Waveform. The continuous MFSK waveform is often used in automotive collision avoidance radar systems where simultaneous range and velocity determination is required. The waveform is constructed by interleaving two stepped sequences of increasing frequency CW waves.

## Pattern methods for antennas, microphones, and arrays

For antenna, microphone, and array System objects, a new `pattern` method replaces the existing `plotResponse` method. In addition, this release introduces two simplified methods for drawing 2-D azimuth and elevation pattern plots: `azimuthPattern` and `elevationPattern`. See, for example, `phased.ULA.pattern`, `phased.ULA.patternAzimuth`, and `phased.ULA.patternElevation`.

This table is a guide for converting your code from using `plotResponse` to using `pattern`. Some of the inputs have changed from input arguments to name-value pair arguments and the converse. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

| plotResponse Inputs | plotResponse Description | pattern Inputs |
|---|---|---|
| `H` argument | Antenna, microphone, or array System object. | `H` argument (no change) |
| `FREQ` argument | Operating frequency. | `FREQ` argument (no change) |
| `V` argument | Propagation speed. This argument is used only for arrays. | `'PropagationSpeed'` name-value pair. This parameter is only used for arrays. |

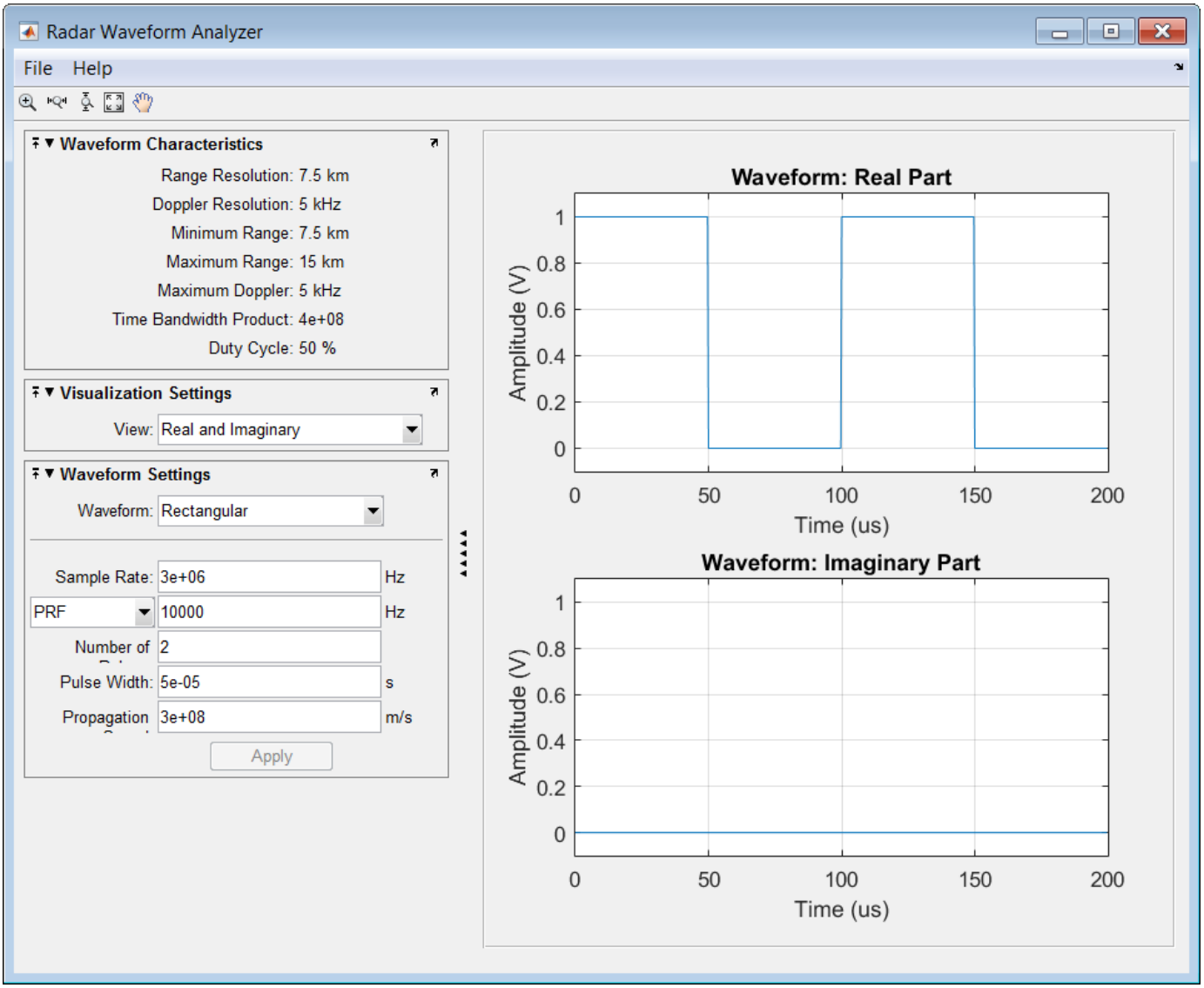
<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.<br><br>Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.<br><br>Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.<br><br>'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

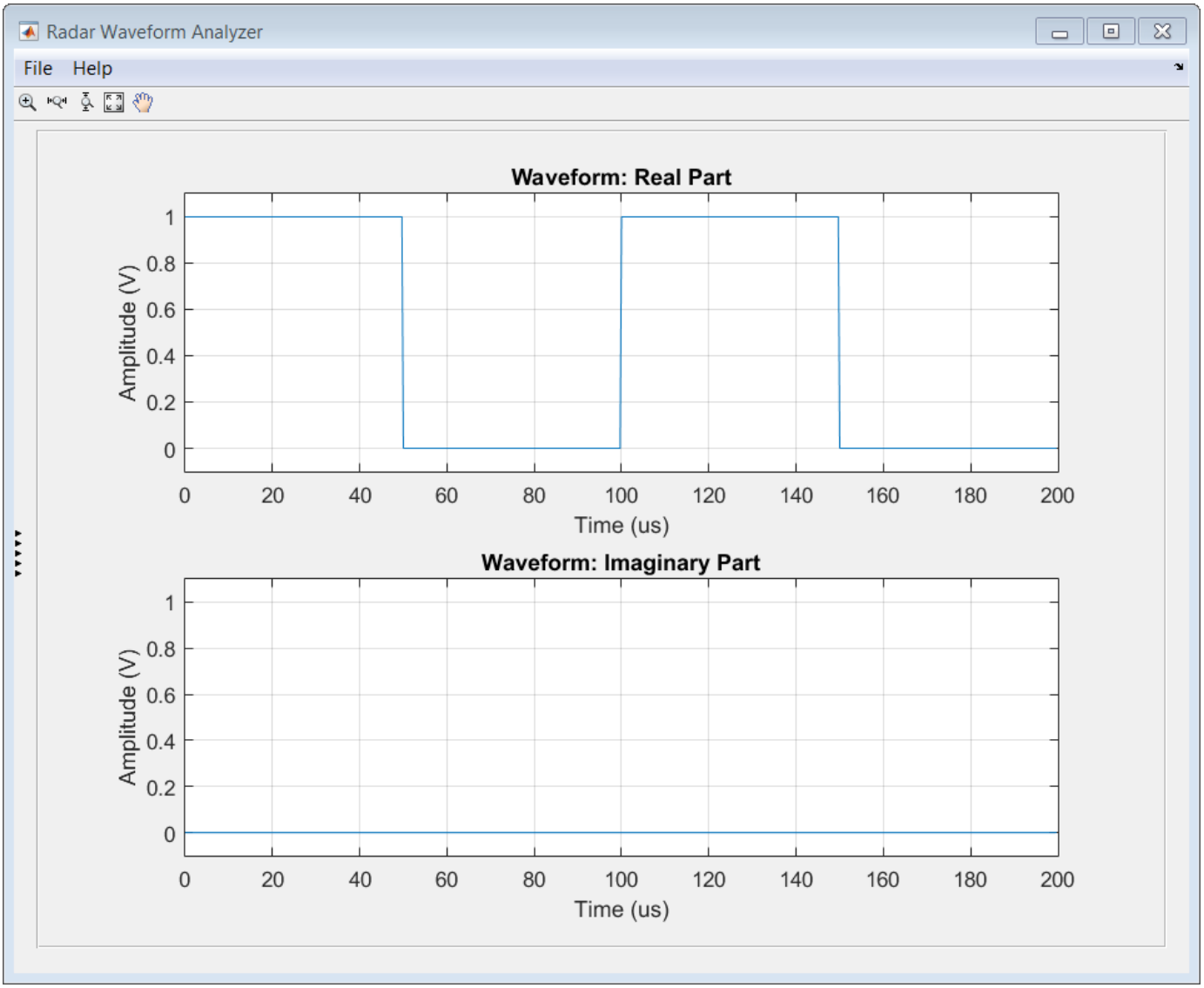
<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.<br><br>The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names<br><br><table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

| plotResponse Inputs | plotResponse Description | pattern Inputs |
|---|---|---|
| `'ElevationAngles'` name-value pair | Elevation angles used to display the antenna or array response. | `EL` argument |
| `'UGrid'` name-value pair | Contains *U* coordinates in *UV*-space. | `AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'` |
| `'VGrid'` name-value pair | Contains *V*-coordinates in *UV*-space. | `EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'` |

## Sensor Array Analyzer and Radar Waveform Analyzer configurable layout

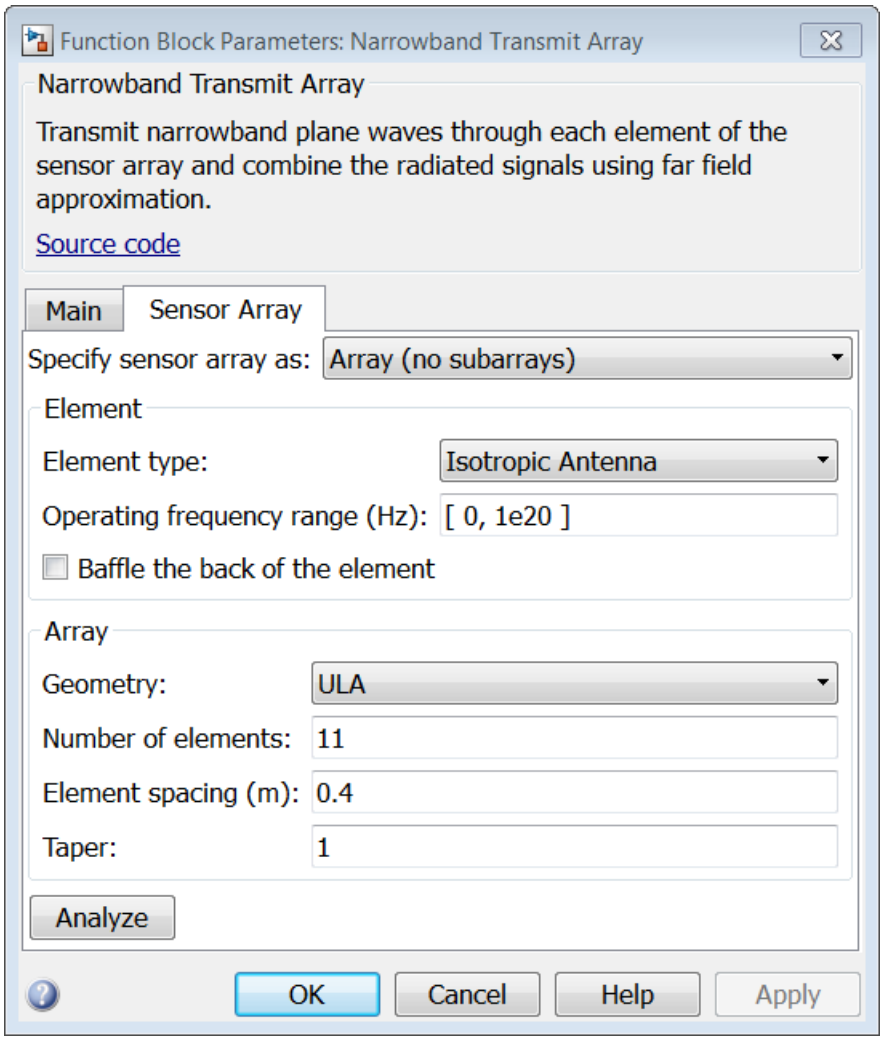
You can now hide, rearrange, and undock the configuration panels in the Sensor Array Analyzer and Radar Waveform Analyzer apps. The content and functionality have not changed. These figures show how you can hide the **Waveform Characteristics**, **Visualization Settings** and **Waveform Settings** panes so that you can focus on the waveform shape.


Radar Waveform Analyzer
File Help
Waveform Characteristics
Range Resolution: 7.5 km
Doppler Resolution: 5 kHz
Minimum Range: 7.5 km
Maximum Range: 15 km
Maximum Doppler: 5 kHz
Time Bandwidth Product: 4e+08
Duty Cycle: 50 %
Visualization Settings
View: Real and Imaginary
Waveform Settings
Waveform: Rectangular
Sample Rate: 3e+06 Hz
PRF 10000 Hz
Number of 2
Pulse Width: 5e-05 s
Propagation 3e+08 m/s
Apply
Waveform: Real Part
Amplitude (V)
Time (us)
Waveform: Imaginary Part
Amplitude (V)
Time (us)

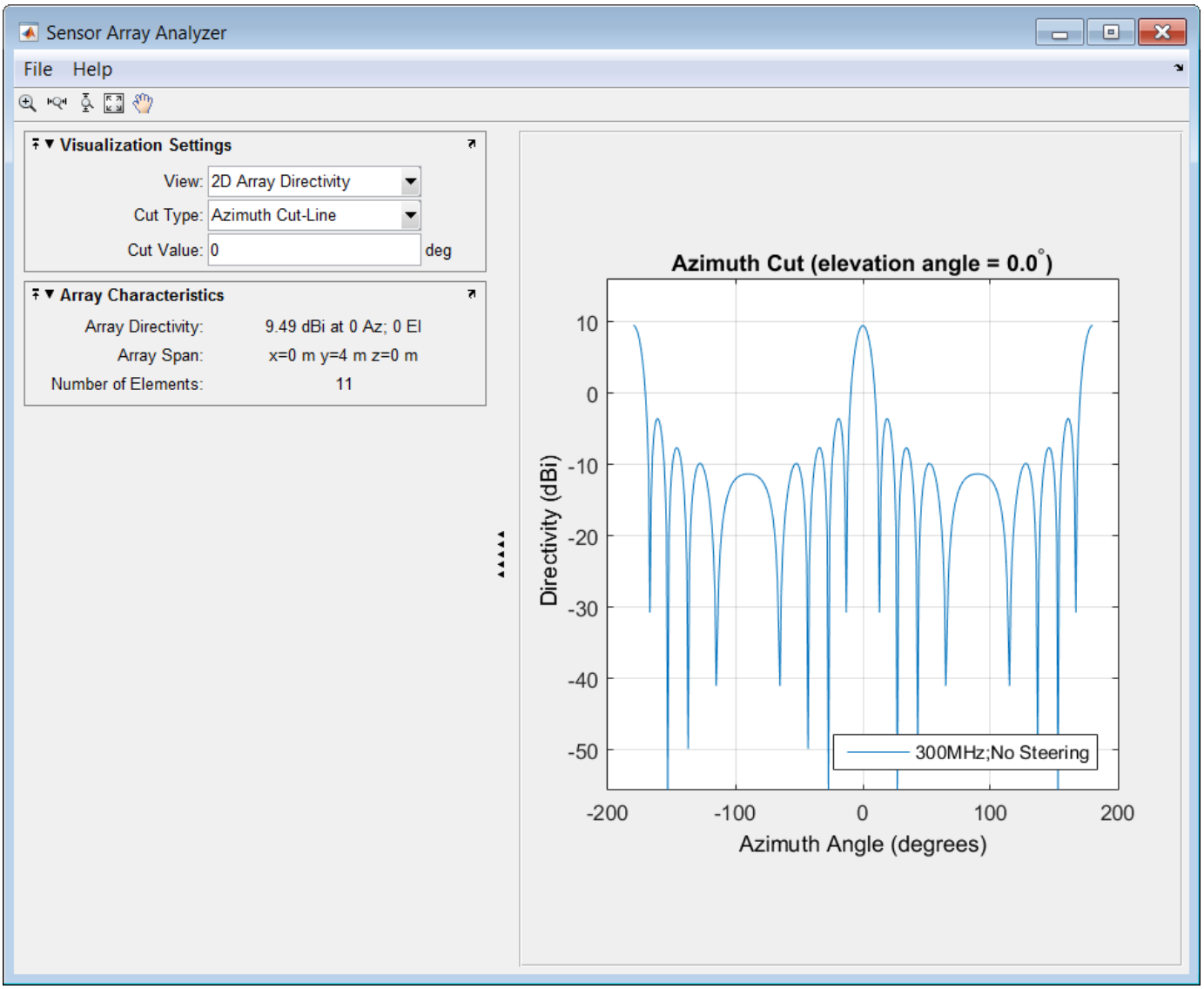



## Launch Sensor Array Analyzer from Tx and Rx blocks

The Narrowband Transmit Array and Narrowband Receive Array blocks let you display and analyze properties of the arrays that you use in your model. You can display an array's shape, its 2-D and 3-D directivities, and its grating lobe structure. To use this feature, on the **Sensor Array** panel of either block, click **Analyze**. As an example, this Narrowband Transmit Array block dialog models an 11-element ULA with elements spaced 0.4 meters apart with an operating frequency of 300 MHz.




Click **Analyze** to open the Sensor Array Analyzer app and show the array directivity.




## Improvements for creating System objects

- Number of allowable code generation inputs increased to 32
- `isInputSizeLockedImpl` method for specifying whether the input port dimensions are locked
- `matlab.system.display.Action` class, used in the `getPropertyGroupsImpl` method, to define a MATLAB System block button that can call a System object method
- `getSimulateUsingImpl` and `showSimulateUsingImpl` methods to set the value of the `SimulateUsing` parameter and specify whether to show the `SimulateUsing` parameter in the MATLAB System block

## Changed default direction of UCA array normal in Sensor Array Analyzer App

In the Sensor Array Analyzer app, the default direction of the array normal vector for the uniform circular array has changed. The array normal is now along the *z*-axis of the local coordinate system. Previously, the array normal was along the *x*-axis.

## Compatibility Considerations

If you set the **Array Type** parameter to `Uniform Circular`, the app displays a different array geometry and different array patterns.

## Functionality being removed or changed

| Functionality | What Happens When You Use This Functionality? | Use This Instead | Compatibility Considerations |
|---|---|---|---|
| `plotResponse` methods for all antenna, microphone, and array System objects | Still runs | `pattern` | Replace all instances of `plotResponse` with `pattern` or consider using the new simplified methods `patternAzimuth` and `patternElevation`. See “Pattern methods for antennas, microphones, and arrays” on page 12-3 for conversion information. |
| `Sensor Array Analyzer` app | Still runs | | Users will see different a array geometry and different array patterns for uniform circular arrays. |

# R2014b

## Simulink blocks for phased array system design

This release adds phased array processing to Simulink. You can use Simulink to model from end-to-end the transmission, propagation, reception, and detection of signals from antenna and microphone arrays. Over forty blocks are available in 7 libraries with these capabilities.

| Phased Array Library | Purpose |
|---|---|
| Beamforming | Conventional and adaptive beamformers, such as LCMV and MVDR |
| Detection | Detection algorithms, including matched filtering and CFAR |
| Direction of Arrival | DOA algorithms, including beamscan, MUSIC, and ESPRIT |
| Environment | Clutter models, jammers, and target models |
| STAP | Space-time adaptive processing |
| Transmitters and Receivers | Arrays for transmitting and receiving |
| Waveforms | Signal waveforms, such as Linear FM, FMCW, and rectangular |

## Directivity of antennas, microphones, and phased arrays

This release adds directivity methods to each antenna and microphone System object and each array, replicated subarray, and partitioned array System object. Directivity is a useful measure of performance for antenna and microphone elements and arrays of elements. Using these methods, you can calculate the directivity values for different directions and frequencies. For examples using the new methods for several System objects, see these pages.

| Phased Array System object | Directivity Method |
|---|---|
| Cosine Antenna Element | `phased.CosineAntennaElement.directivity` |
| Uniform Linear Array | `phased.ULA.directivity` |
| Partitioned Array | `phased.PartitionedArray.directivity` |

## Remove NoiseBandwidth property from phased.ReceiverPreamp System object

The `NoiseBandwidth` property of the `phased.ReceiverPreamp` System object is obsolete and may be removed in a future release. This property is not used in any computation. As of this release, a warning message will be displayed when `phased.ReceiverPreamp` is invoked with the `NoiseBandwidth` property specified. To suppress this message, type `warning off phased:system:System:NoiseBandwidthWarning` at the command line at any time. Alternatively, you can include this command in your `startup.m` script or other scripts that use `phased.ReceiverPreamp`.

## Compatibility Considerations

To avoid future incompatibility, remove all occurrences of the `NoiseBandwidth` property.

## Property value name change for phased.RangeDopplerResponse System object

In the `RangeMethod` property of the `phased.RangeDopplerResponse` System object, the `'Dechirp'` value has been renamed to `'FFT'` value. This change is a name change only. The `'Dechirp'` value name will be removed in a future release.

## Compatibility Considerations

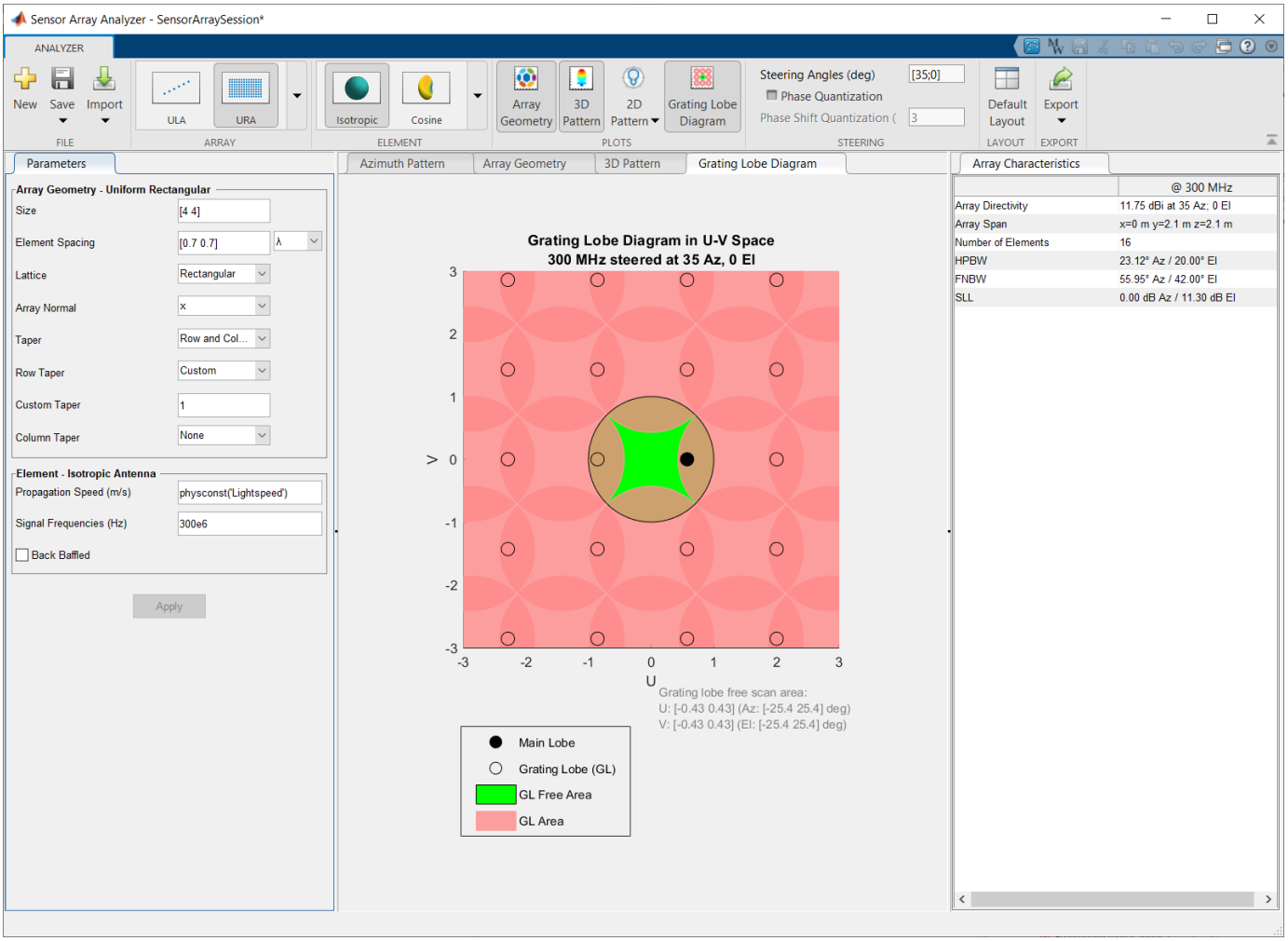
When using the `RangeMethod` property in your code, change all occurrences of `'Dechirp'` to `'FFT'`.

# R2014a

## 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.



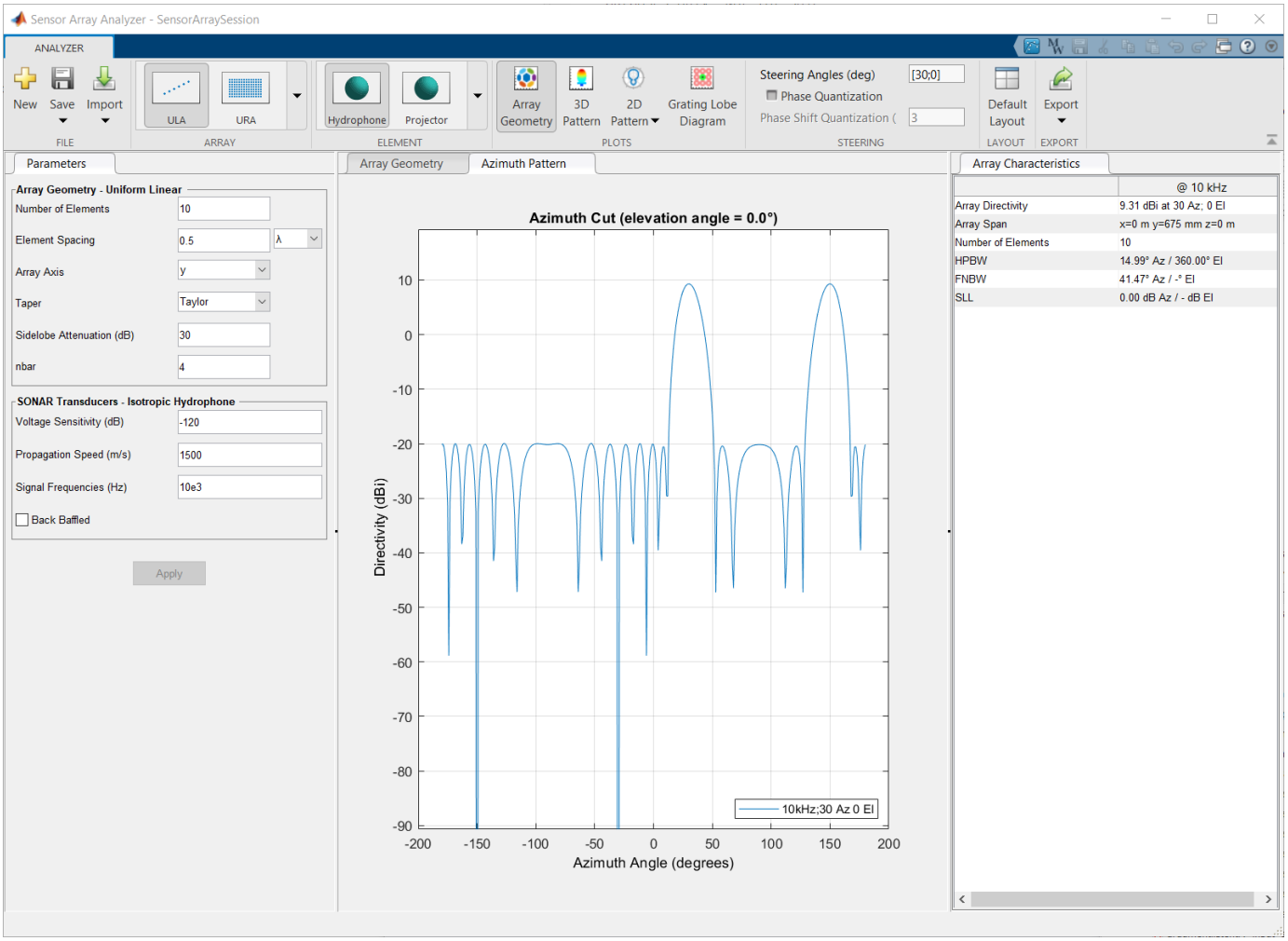

## Visualization of element and array directivity

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.




## 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 14-2.
- You can plot grating lobe diagrams for several types of arrays, as described in “Grating lobe diagrams” on page 14-2.
- 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

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

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

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**

**Compatibility Considerations**

## 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

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

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

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

When you use the `plotResponse` method to return a 3-D 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

## 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<br>phased.CustomMicrophoneElement<br>phased.IsotropicAntennaElement<br>phased.OmnidirectionalMicrophoneElement<br>phased.CrossedDipoleAntennaElement (new)<br>phased.ShortDipoleAntennaElement (new) | | isPolarizationCapable<br>plotResponse<br>step |
| | phased.CustomAntennaElement | SpecifyPolarizationPattern<br>HorizontalMagnitudePattern<br>HorizontalPhasePattern<br>VerticalMagnitudePattern<br>VerticalPhasePattern | |
| Array Geometries and Analysis | phased.ConformalArray<br>phased.ULA<br>phased.URA<br>phased.PartitionedArray<br>phased.ReplicatedSubarray | | isPolarizationCapable<br>plotResponse<br>step |
| | phased.SteeringVector<br>phased.ArrayResponse | EnablePolarization | |
| Signal Radiation and Collection | phased.Collector<br>phased.WidebandCollector<br>phased.Radiator | EnablePolarization | step |
| Environment and Target Models | phased.RadarTarget | EnablePolarization<br>Mode<br>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 |
| `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, 2D, 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

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

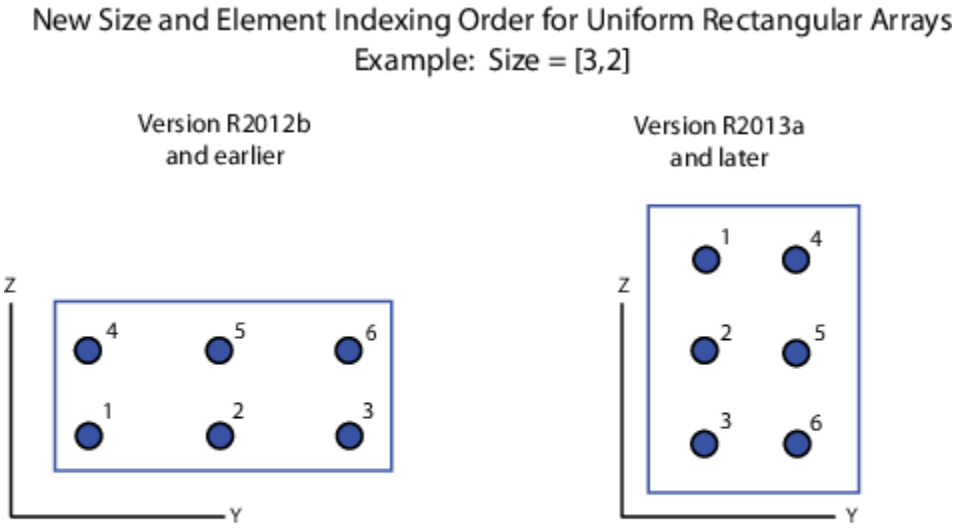
## Compatibility Considerations

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

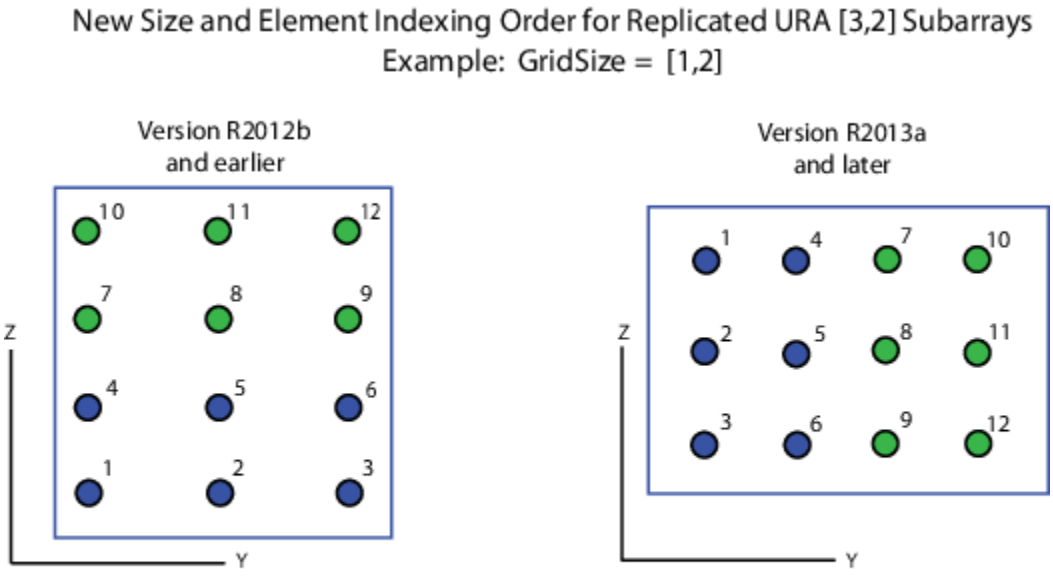
## Element indexing pattern change for phased.URA and phased.ReplicatedSubarray

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.




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**

**Compatibility Considerations**

## 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

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

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

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**

**Compatibility Considerations**

## Replicated Subarrays and Partitioned Array Apertures

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.

## 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.

## 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 an example, look at Antenna Radiation Pattern in U/V Coordinates. For background information about the coordinate systems, see Spherical Coordinates.

## Multiple-Beam Beamformers

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

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

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**

**Compatibility Considerations**

## 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`

## 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

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

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

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 for more information.

## Conversion of Error and Warning Message Identifiers

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**

## 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