Communications Toolbox™ Release Notes

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Communications Toolbox™ Release Notes

Trademarks

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

Patents

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

Contents

Summary by Version

This table provides quick access to what's new in each version. For clarification, see "Using Release Notes" on page 2 .

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V4.6 (R2010b)	Yes Details	No	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation
V4.5 (R2010a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.4 (R2009b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.3 (R2009a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.2 (R2008b)	Yes Details	No	Bug Reports Includes fixes	No
V4.1 (R2008a)	Yes Details	No	Bug Reports Includes fixes	No
V4.0 (R2007b)	Yes Details	No	Bug Reports Includes fixes	No
V3.5 (R2007a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V3.4 (R2006b)	Yes Details	Yes Summary	Bug Reports	No
V3.3 (R2006a)	Yes Details	No	Bug Reports Includes fixes	No
V3.2 (R14SP3)	Yes Details	No	Bug Reports	No

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
V3.1 (R14SP2)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V3.0.1 (R14SP1)	Yes Details	No	Fixed bugs	No
V3.0 (R14)	Yes Details	Yes Summary	Fixed bugs	No
V2.1 (R13)	Yes Details	Yes Summary	Fixed bugs and known problems	No
V2.0 (R12)	Yes Details	Yes Summary	Fixed bugs	No

Using Release Notes

Use release notes when upgrading to a newer version to learn about:

- New features
- Changes
- Potential impact on your existing files and practices

Review the release notes for other MathWorks® products required for this product (for example, MATLAB® or Simulink®). Determine if enhancements, bugs, or compatibility considerations in other products impact you.

If you are upgrading from a software version other than the most recent one, review the current release notes and all interim versions. For example, when you upgrade from V1.0 to V1.2, review the release notes for V1.1 and V1.2.

What Is in the Release Notes

New Features and Changes

- New functionality

- Changes to existing functionality

Version Compatibility Considerations

When a new feature or change introduces a reported incompatibility between versions, the **Compatibility Considerations** subsection explains the impact.

Compatibility issues reported after the product release appear under Bug Reports at the MathWorks Web site. Bug fixes can sometimes result in incompatibilities, so review the fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

MathWorks offers a user-searchable Bug Reports database so you can view Bug Reports. The development team updates this database at release time and as more information becomes available. Bug Reports include provisions for any known workarounds or file replacements. Information is available for bugs existing in or fixed in Release 14SP2 or later. Information is not available for all bugs in earlier releases.

Access Bug Reports using your MathWorks Account.

Version 4.6 (R2010b) Communications Toolbox Software

This table summarizes what's new in Version 4.6 (R2010b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

Embedded MATLAB Support for Communications Toolbox Functions

Embedded MATLAB® supports the generation of embeddable C code for the following Communications Toolbox™ functions:

- `bi2de`
- `de2bi`
- `istrellis`
- `poly2trellis`
- `rcosfir`

The generated C code meets the strict memory and data type requirements of embedded target environments.

Support for this functionality requires Communications Toolbox and Communications Blockset™ software licenses. Some Embedded MATLAB features also require additional product licenses. See "Which Embedded MATLAB Feature to Use" for a comprehensive list.

For a comprehensive list of Embedded MATLAB functions that support C code generation, see "Embedded MATLAB Function Library Reference".

For additional information, see “Working with Embedded MATLAB Subset in Communications Toolbox”in *Communications Toolbox User's Guide*.

Version 4.5 (R2010a) Communications Toolbox Software

This table summarizes what's new in Version 4.5 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes Summary	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

Error Rate Test Console Enhancements

The Error Rate Test Console now contains:

- Plotting enhancements,
- A new stop simulation option
- Simultaneous specification of sweep values
- Descriptions for test probes

Surface Plotting Capabilities

The `results` object supports a new surface plotting method. The `surf(r)` method creates a 3-D, color, surface plot based on the results available in the `results` object. The following items define the surface plot:

- The test point you specify using the `TestPoint` property of the `results` object

- The test metric currently you specify in the `Metric` property of the `results` object

For more information see `testconsole.Results`.

New Stop Simulation Option

The `SimulationLimitOption` property supports a new stop option: 'Number of errors and transmissions'.

The simulation runs for the number of transmissions you specify. Through these runs, the toolbox ensures that a minimum number of errors occur. These multiple runs also provide a long enough observation time for both deep fades and high gain phases of the channel to appear. For more information, see `commtest.ErrorRate`.

Simultaneous Specification of Sweep Values

The `setTestParameterSweepValues` method now supports simultaneous specification of sweep values, allowing you to specify the sweep values for multiple registered test parameters at one time. For more information, see the `setTestParameterSweepValues` method section of the `commtest.ErrorRate` help page.

Probe Descriptions

You can add a description for a probe in the system under test. You can read the description for the probe using `getTestProbeDescription` method of the Error Rate Test Console. This enhancement allows you to review the probe's use without having to access the code for the system under test. For more information, see "Registering Test Probes" in the *Communications Toolbox User's Guide*.

Parsing of Results

The `results` object has two new methods, setParsingValues and getParsingValues. which are relevant for parsing and plotting data.

Use these methods to specify single sweep values for test parameters that differ from the ones in TestParameter1 and TestParameter2. The results object returns data values or plots that correspond to test parameters you

specify in TestParameter1 and TestParameter2 and single sweep values you specify using the setParsingValues method for all additional test parameters. The parsing values default to the first value in the sweep vector of each test parameter.

You display the current parsing values by calling the getParsingValues method of the results object.

For more information, see: "Parsing and Plotting Results for Multiple Parameter Simulations", `testconsole.Results` and `commtest.ErrorRate`.

Compatibility Considerations

Loading a test console object defined in a previous version of the Error Rate Test Console software results in a warning message. The warning instructs you to resave the test console object. To prevent this warning from appearing, resave the test console object using the latest version of the Error Rate Test Console software.

Surface Plotting Capabilities

The `results` object supports a new surface plotting method. The `surf(r)` method creates a 3-D, color, surface plot based on the results available in the `results` object. The following items define the surface plot:

- The test point you specify using the `TestPoint` property of the `results` object
- The test metric currently you specify in the `Metric` property of the `results` object

For more information see `testconsole.Results`.

biterr Function Supports Logical Input Arguments

The `biterr` function now accepts logical input arguments. For more information, including examples, see the `biterr` help page in the Communications Toolbox Reference Guide.

Demos

This release contains a new demo, Passband Modulation with Adjacent Channel Interference, which shows how to use baseband modulators and demodulators with frequency upconversion and downconversion to simulate passband communication systems.

Version 4.4 (R2009b) Communications Toolbox Software

This table summarizes what's new in Version 4.4 (R2009b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes Summary	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are:

Adjacent Channel Power Ratio (ACPR) Measurements

Adjacent channel power ratio (ACPR) calculations characterize spectral regrowth, caused by amplifier nonlinearity, in a communications system component, such as a modulator or an analog front end. These calculations determine the likelihood that a given system causes interference with an adjacent channel.

Many present transmission standards, such as IS-95, CDMA, WCDMA, 802.11, and Bluetooth, contain a definition for ACPR measurements. Most standards define ACPR measurements as the ratio of the average power in the main channel and any adjacent channels. The specific offset frequencies and measurement bandwidths (BWs) you use depends on the specific industry standard you are using. For instance, measurements of CDMA amplifiers

involve two offsets (from the carrier frequency) of 885 kHz and 1.98 MHz, and a measurement BW of 30 KHz.

For more information, see the `commmeasure.ACPR` help page or Overview of ACPR Measurement Tutorial in the Communications Toolbox User's Guide.

New EVM Normalization Options

`commmeasure.EVM` now supports three normalization options. You can normalize measurements according to the average power of the reference signal, average constellation power, or peak constellation power. This enhancement provides you with the flexibility to use EVM normalization with the different definitions of EVM measurements that appear in various industry standards. Typically, the variations in these standards pertain to the normalization option.

For more information, see the `commmeasure.EVM` help page.

Error Rate Test Console

Communications Toolbox contains a command-line approach for simulating error rate in a communications system. The error rate test console runs simulation on a user-defined communications system to obtain error rate analysis.

If you also have a user license for the Parallel Computing Toolbox software, the error rate test console reduces simulation time by automatically distributing the work load among the number of available processors.

For more information, see the `commtest.ErrorRate` help page, Running Simulations Using the Error Rate Test Console in the Communications Toolbox *Getting Started Guide*, or Error Rate Test Console in the Communications Toolbox User's Guide.

Channel Objects Support Parallel Computing Toolbox

`rayleighchan`, `ricianchan`, and `mimochan` now support the Parallel Computing Toolbox™ software. Using the default MATLAB random stream algorithm, channel objects can generate independent channels on different workers. Since `rayleighchan`, `ricianchan`, and `mimochan` can generate

independent channels on each worker, you can use the Parallel Computing Toolbox software to run simulations on multiple workers, reducing simulation time.

Compatibility Considerations

`rayleighchan`, `ricianchan`, and `mimochan` now use the default MATLAB random stream. Software versions before the 2009b release use the V5 RANDN (Ziggurat) algorithm to generate channel path gains. The default random stream is more robust for use with Parallel Computing Toolbox software and can generate channel path gain values that are the statistical equivalent to the V5 RANDN (Ziggurat) algorithm.

`rayleighchan`, `ricianchan`, and `mimochan` do not support `reset(h,s)`, where *h* represents a channel and *s* represents the new channel state. To obtain the random number generation functionality before the 2009b release, including `reset(h,s)` support, use `legacychannelsim`. You can not use channel objects with Parallel Computing Toolbox software to run simulations in legacy mode.

New Demos

The following demos are new or updated for this release:

- Two new demos show how to simulate 3GPP EGPRS2 and IEEE 802.11b communications systems using the Error Rate Test Console. These demos can be run on multiple workers if you have a user license for the Parallel Computing Toolbox software.
- The Raised Cosine demo now uses the filter design object `fdesign.pulseshaping`.

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When you use the Function or Element?	Use This Instead	Compatibility Considerations
`seqgen.pn`	Warns	`commsrc.pn`	Use `commscrc.pn` to create a PN sequence generator object.
`rcosfir`	Still runs	`fdesign.pulseshaping`	Use `fdesign.pulseshaping` to design raised cosine finite (FIR) impulse response filter
`rcosflt`	Still runs	`fdesign.pulseshaping`	Use `fdesign.pulseshaping` to design a filter input signal using raised cosine filter. Does not support IIR.
`rcosiir`	Still runs	N\A	Do not use.
`rcosine`	Still runs	`fdesign.pulseshaping`	Use `fdesign.pulseshaping` to design a design a raised cosine filter. Does not support IIR.
`randint`	Still runs	`randi`	Use `Randi` to generate matrix of uniformly distributed random integers
`reset(h,s)` for `rayleighchan`	Errors	N\A	Do not use. See `legacychannelsim`
`reset(h,s)` for `ricianchan`	Errors	N\A	Do not use. See `legacychannelsim`
`reset(h,s)` for `mimochan`	Errors	N\A	Do not use. See `legacychannelsim`
`state` parameter for `awgn`	Still runs	*s*, which is a random stream handle	None.

Function or Function Element Name	What Happens When you use the Function or Element?	Use This Instead	Compatibility Considerations
`state` parameter for `wgn`	Still runs	*s*, which is a random stream handle	None.
`state` parameter for `bsc`	Still runs	*s*, which is a random stream handle	None.
`state` parameter for `randerr`	Still runs	*s*, which is a random stream handle	None.
`state` parameter for `randsrc`	Still runs	*s*, which is a random stream handle	None.

Version 4.3 (R2009a) Communications Toolbox Software

This table summarizes what's new in Version 4.3 (R2009a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are

Bell-Shape Doppler Object

The doppler.bell function creates a bell-shape Doppler spectrum object. You can use this object for the `DoplerSpectrum` property of a channel object for the `rayleighchan` function, `ricianchan` function, or the `mimochan` function. For more information, refer to the `doppler.bell` help page.

Scatter Plot

Communications Toolbox supports the creation of a scatter plot Graphical User Interface (GUI) that displays measurement and scatter plot results in the same figure. The scatter plot feature is part of the `commscope` package. Users can create the scatter plot object in two ways: using a default object or by defining parameter-value pairs. To access the scatter plot feature, type `commscope.ScatterPlot` at the MATLAB command line. For more information, see the `commscope.ScatterPlot` help page.

EVM and MER Measurements

Communications Toolbox can perform Error Vector Magnitude (EVM) and Modulation Error Ratio (MER) measurements. EVM is a measurement of demodulator performance in the presence of impairments. MER is a measure of the signal-to-noise ratio (SNR) in a digital modulation application. These types of measurements are useful for determining system performance in communications applications. For example, determining if an EDGE system conforms to the 3GPP radio transmission standards requires accurate RMS, EVM, Peak EVM, and 95th percentile for the EVM measurements.

Users can create an EVM object in two ways: using a default object or by defining parameter-value pairs. As defined by the 3GPP standard, the unit of measure for RMS, Maximum, and Percentile EVM measurements is percentile (%). To access the EVM object, type `commmeasure.EVM` at the MATLAB command line. For more information, see the `commmeasure.EVM` help page.

The MER object is part of the `commmeasure` package. As defined by the DVB standard, the unit of measure for MER is decibels (dB). For consistency, the unit of measure for Minimum MER and Percentile MER measurements is also in decibels. To access this object, type `commmeasure.MER` at the MATLAB command line. For more information, see the `commmeasure.MER` help page.

commsrc package now supports PN sequence generation

In order to make all of the communications sources available together, the sequence generator functionality is now part of the `commsrc` package. This object provides the same characteristics and behavior as in previous Toolbox versions. As a result, the `seqgen.pn` function is obsolete and will be removed in a future release.

Compatibility Consideration

Note MathWorks will remove the `seqgen` function from a future version of the Communications Toolbox product. While the product still supports this function, you should use `commsrc.pn` instead.

MIMO channel simulation support

The new MIMO channel object supports multiple-input multiple output simulations. You can specify correlated or uncorrelated fading between channels. For more information, see the `mimochan` help page or the IEEE® 802.11b WLAN Physical Layer demo.

Version 4.2 (R2008b) Communications Toolbox Software

This table summarizes what's new in Version 4.2 (R2008b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are

Pattern Generator

Communications Toolbox software now includes Pattern Generator object. You can use this object to generate NRZ or RZ signals and inject jitter, which enables stress tests for high speed communications systems. For more information, refer to the `commsrc.pattern` help page.

EyeScope Compare Measurement Results View

EyeScope now supports the ability to import multiple eye diagram objects. Additionally, EyeScope now contains a **View** menu, which you can use to compare measurement results from multiple eye diagram objects.

New Demos

New demos and demos containing updates for this release:

- New `Multipath Fading Channel Modeling` demo.

- Updates to `Eye Diagram Measurements` demo include use of pattern generator, importing multiple eye diagram objects, and comparing measurement results.

Version 4.1 (R2008a) Communications Toolbox Software

This table summarizes what's new in Version 4.1 (R2008a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are

Cyclic Redundancy Checking (CRC)

New CRC generator and detector objects have been added to the Communications Toolbox software.

Eye Diagram Measurements

Communications Toolbox product now provides eye diagram measurements, generated from the histogram data of eye diagram objects. This feature will be helpful for signal integrity analysis, a major focus of backplane and optical communications development.

You can obtain the following measurements on an eye diagram:

- Eye Crossing Times
- Eye Crossing Amplitude

- Eye Delay
- Eye Level
- Eye Amplitude
- Eye Height
- Eye Width
- Vertical Eye Opening
- Eye Crossing Percentage
- Eye SNR
- Quality Factor
- Random Jitter
- Deterministic Jitter
- Total Jitter
- RMS Jitter
- Peak-to-Peak Jitter
- Horizontal Eye Opening
- Eye Rise Time
- Eye Fall Time

EyeScope Functionality

EyeScope can be used to examine eye diagram results in a user-friendly, graphical environment. EyeScope shows both the eye diagram figure and measurement results in a unified GUI, providing a more efficient means for viewing results.

New BCH and Reed Solomon Objects

Communications Toolbox software now contains BCH and Reed Solomon objects. These objects extend the capabilities of existing BCH and Reed Solomon functions by enabling punctures and erasure coding. Refer to the following reference pages for more information:

- `bchdec`
- `bchenc`
- `fec.rsdec`
- `fec.rsenc`

New Demos

New demos have been added for `spatial multiplexing` and `Eye Diagram Measurements`.

- The `spatial multiplexing` demo illustrates how to perform `spatial multiplexing` using Successive Interference Cancellation (SIC) detection schemes.
- The `Eye Diagram Measurements` demo illustrates how to use the commscope.eyediagram object to perform eye diagram measurements on simulated signals.

Version 4.0 (R2007b) Communications Toolbox Software

This table summarizes what's new in Version 4.0 (R2007b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes.	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are

Bi-Gaussian Doppler and Asymmetrical Jakes Doppler Spectra Objects Added

The `doppler.bigaussian` function creates a bivariate Gaussian Doppler spectrum object, which is used in the `DopplerSpectrum` property of a channel object.

Channel Objects and Channel Visualization Tool Enhanced

Channel objects and channel visualization tool are enhanced to support different Doppler spectra per path.

ricianchan Object Enhanced

All paths now have the option of being Rician-faded (previously only the first path of a Rician multipath fading channel could be Rician-faded).

Each path can be assigned different values for the `K` and `fdLOS` parameters.

See the `ricianchan` reference page for details.

Demos Modeling Various Channels Added

New demos modeling COST 207, GSM/EDGE, and HF are added.

These can be accessed through the demos pane of the Help browser, or by typing `chandemo_cost207` and `chandemo_hf` at the command line.

Eye Diagram Object Added

New eye diagram object is added, that uses color to convey how often a trace traverses a point.

Modem Objects Added

New PAM, OQPSK, DPSK, MSK, and General QAM modem objects are added.

Theoretical BERs Added to BERTool and Enhanced in bercoding

Theoretical BERs added to BERTool for hard-decision and soft-decision decoding with AWGN. Existing results in `bercoding` are made more precise.

New stdchan Object

The new `stdchan` object constructs channel objects according to standardized channel models, including COST 207, GSM/EDGE, 3GPP, and ITU-R.

bchenc Enhanced

The `bchenc` function now runs faster, especially for longer codeword lengths.

Version 3.5 (R2007a) Communications Toolbox Software

This table summarizes what's new in Version 3.5 (R2007a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes Summary	Bug Reports Includes fixes.	No

New features and changes introduced in this version are

Added Support for Different Doppler Spectra to Channel Objects

A package of Doppler classes is added. These classes are used to instantiate Doppler objects to be used with the Rayleigh and Rician channel objects:

- `doppler.flat`
- `doppler.gaussian`
- `doppler.jakes`

- `doppler.rjakes`
- `doppler.rounded`

See individual reference pages for details.

Theoretical Results Refined for `berawgn`, `berfading`, and BERTool

Many of the previous theoretical results are now more accurate.

The functions `berawgn` and `berfading` can now return the symbol error rate in addition to the bit error rate.

Several new theoretical results have been added.

Compatibility Considerations

The results from these functions may be different from those of previous releases as they are now more accurate.

Various Enhancements Made to BERTool

When performing Monte Carlo simulations using BERTool, the BER plot is now updated every time a new data point is calculated. This allows the user to see whether the calculation is on the right track, and possibly start using the results while the calculation is in progress.

Theoretical BER plots are now shown to be one of exact, approximate, lower bounds, or upper bounds.

Theoretical error control coding BER results for Hamming, Golay, and Reed Solomon codes are now available in the BERTool.

See "BERTool: A Bit Error Rate Analysis GUI" in the *Communications Toolbox User's Guide* for details.

New Demo on MIMO Systems Showcasing Space-Time Block Coding

A new demo on MIMO systems is added, illustrating orthogonal space-time block coding. You can open this demo by typing `showdemo('introMIMOSystems')` at the command line.

New PN Sequence Generator Object

The object `seqgen.pn` produces a pseudorandom noise sequence using a linear-feedback shift register that is implemented using a simple shift register generator (SSRG, or Fibonacci). See reference page for details.

New `dvbs2ldpc` Function

The function `dvbs2ldpc` returns the parity-check matrix of an LDPC code from the DVB-S.2 standard.

New `finddelay` Function

The function `finddelay` estimates delays between signals.

New `alignsignals` Function

The function `alignsignals` aligns two signals by delaying the earlier signal.

Enhanced Speed of `rsdec` and `bchdec` Functions

The functions `rsdec` and `bchdec` are enhanced to run significantly faster.

Version 3.4 (R2006b) Communications Toolbox Software

This table summarizes what's new in Version 3.4 (R2006b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes Summary	Bug Reports	No

New features and changes introduced in this version are

Theoretical BER Results Added

Theoretical error control coding BER results for Hamming, Golay, and Reed Solomon codes are now available in the function `bercoding`.

Bitwise Soft-Decision Outputs for PSK and QAM Demodulators

Bitwise soft-decision outputs are enabled for the PSK and QAM demodulation through the use of new PSK and QAM modem objects. See "Object-Based PSK and QAM Modulation and Demodulation" on page 30.

Added LDPC Encoder and Decoder Objects

The objects `fec.ldpcenc` and `fec.ldpcdec` respectively encode and decode LDPC codes.

Line-of-Sight Doppler Shift Added to Rician Channel

The property `DirectPathDopplerShift`, which specifies the maximum Doppler shift of the line-of-sight path, is added to the Rician channel object.

Object-Based PSK and QAM Modulation and Demodulation

PSK and QAM modulation and demodulation is now done using the new classes `modem.pskmod`, `modem.pskdemod`, `modem.qammod`, and `modem.qamdemod`.

Compatibility Considerations

See "Using Modem Objects" and individual reference pages for details.

QAM and PSK Modulation and Demodulation Functions Obsoleted

The functions `pskmod`, `pskdemod`, `qammod`, and `qamdemod` are obsolete and may be removed in the future.

Compatibility Considerations

Use the new object based PSK and QAM modulation and demodulation objects instead. See "Object-Based PSK and QAM Modulation and Demodulation" on page 30.

Version 3.3 (R2006a) Communications Toolbox Software

This table summarizes what's new in Version 3.3 (R2006a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes.	No

New features and changes introduced in this version are

convenc and vitdec Updated with Puncturing and Erasing

The function `convenc` is updated with puncturing capabilities. The function `vitdec` now decodes codewords with punctures and erasures. Note that their function syntax have also changed.

Enhanced pamdemod, pskdemod, and qamdemod Functions

The `pamdemod`, `pskdemod`, and `qamdemod` functions are enhanced to run significantly faster.

Version 3.2 (R14SP3) Communications Toolbox Software

This table summarizes what's new in Version 3.2 (R14SP3):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports	No

New features and changes introduced in this version are

Added function bchnumerr

`bchnumerr` returns all the possible combinations of message lengths and number of correctable errors for a BCH code of given length.

Speed increase in bchgenpoly, bchenc, and bchdec functions

`bchgenpoly`, `bchenc`, and `bchdec` function have been enhanced to run more rapidly.

Version 3.1 (R14SP2Communications Toolbox Software

This table summarizes what's new in Version 3.1 (R14SP2):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes.	No

New features and changes introduced in this version are

Channel Visualization Tool

A new channel visualization tool allows you to plot various channel characteristics.

Improved Rayleigh Fading Channel

Increased the signal processing speed of the Rayleigh Fading channel, `rayleighchan`, by up to a factor of two.

Gray Coding Functionality

Added the functions `bin2gray` and `gray2bin` to convert between Gray decoded and encoded integers.

Added Gray symbol ordering to the functions `pskmod`, `pammod`, `dpskmod`, `qammod`, `fskmod`, `pskdemod`, `pamdemod`, `dpskdemod`, `qamdemod`, and `fskdemod`.

Rician Channel Enhancement to the BERTool

The `bertool` now has theoretical BER results for a Rician channel.

gfrank

Compatibility Considerations

The function `gfrank` now returns `0`, instead of `[]`, on a zero matrix input.

encode, decode, and quantiz

Compatibility Considerations

The outputs of the `encode`, `decode`, and `quantiz` functions now match the input vector's orientation.

Version 3.0.1 (R14SP1) Communications Toolbox Software

This table summarizes what's new in Version 3.0.1 (R14SP1):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Fixed bugs	No

New features and changes introduced in this version are

Rician Channel BER Calculations

The BERTool is enhanced to allow for Rician channel BER calculations. For details, see Available Sets of Theoretical BER Data in the Communications Toolbox documentation.

berfading Updated for Rician Channel

`berfading` is enhanced to return the BER of BPSK over uncoded flat Rician fading channels. For details, see the Communications Toolbox documentation for `berfading`.

New Adaptive Equalization Demo

A new demo illustrates adaptive equalization using Embedded MATLAB. To open the demo, type `equalizer_eml` at the MATLAB command line.

Version 3.0 (R14) Communications Toolbox Software

This table summarizes what's new in Version 3.0 (R14):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Fixed bugs	No

New features and changes introduced in this version are

Bit Error Rate Analysis GUI

Communications Toolbox has a graphical user interface (GUI) called BERTool that helps you analyze communication systems' bit error rate (BER) performance. To invoke the GUI, type

```
bertool
```

in the MATLAB Command Window.

Performance Evaluation

The functions in the table below enable you to measure or visualize the bit error rate performance of a communication system.

Function	Purpose
`berawgn`	Error probability for uncoded AWGN channels
`bercoding`	Error probability for coded AWGN channels
`berconfint`	BER and confidence interval of Monte Carlo simulation
`berfading`	Error probability for Rayleigh fading channels
`berfit`	Fit a curve to nonsmooth empirical BER data
`bersync`	Bit error rate for imperfect synchronization
`distspec`	Compute the distance spectrum of a convolutional code
`semianalytic`	Calculate bit error rate using the semianalytic technique

Equalizers

The functions in the table below enable you to equalize a signal using a linear equalizer, a decision feedback equalizer, or a maximum-likelihood sequence estimation equalizer based on the Viterbi algorithm.

Function	Purpose
`cma`	Construct a constant modulus algorithm (CMA) object
`dfe`	Construct a decision feedback equalizer object
`equalize`	Equalize a signal using an equalizer object
`lineareq`	Construct a linear equalizer object
`lms`	Construct a least mean square (LMS) adaptive algorithm object
`mlseeq`	Equalize a linearly modulated signal using the Viterbi algorithm
`normlms`	Construct a normalized least mean square (LMS) adaptive algorithm object
`rls`	Construct a recursive least squares (RLS) adaptive algorithm object
`signlms`	Construct a signed least mean square (LMS) adaptive algorithm object
`varlms`	Construct a variable step size least mean square (LMS) adaptive algorithm object

Fading Channels and Binary Symmetric Channel

The functions in the tables below enable you to model a Rayleigh fading channel, Rician fading channel, and binary symmetric channel.

Function	Purpose
`bsc`	Model a binary symmetric channel
`filter` (for channel objects)	Filter signal with channel object
`rayleighchan`	Construct a Rayleigh fading channel object
`reset`	Reset channel object
`ricianchan`	Construct a Rician fading channel object

Interleavers

The functions in the tables below enable you to perform block interleaving and convolutional interleaving, respectively.

Block Interleaving

Function	Purpose
`algdeintrlv`	Restore ordering of symbols, using algebraically derived permutation table
`algintrlv`	Reorder symbols, using algebraically derived permutation table
`deintrlv`	Restore ordering of symbols
`helscandeintrlv`	Restore ordering of symbols in a helical pattern
`helscanintrlv`	Reorder symbols in a helical pattern
`intrlv`	Reorder sequence of symbols
`matdeintrlv`	Restore ordering of symbols by filling a matrix by columns and emptying it by rows
`matintrlv`	Reorder symbols by filling a matrix by rows and emptying it by columns
`randdeintrlv`	Restore ordering of symbols, using a random permutation
`randintrlv`	Reorder symbols, using a random permutation

Convolutional Interleaving

Function	Purpose
`convdeintrlv`	Restore ordering of symbols, using shift registers
`convintrlv`	Permute symbols, using shift registers
`heldeintrlv`	Restore ordering of symbols permuted using `helintrlv`
`helintrlv`	Permute symbols, using a helical array

Convolutional Interleaving (Continued)

Function	Purpose
`muxdeintrlv`	Restore ordering of symbols, using specified shift registers
`muxintrlv`	Permute symbols, using shift registers with specified delays

Huffman Coding

The functions in the table below enable you to perform Huffman coding.

Function	Purpose
`huffmandeco`	Huffman decoder
`huffmandict`	Generate Huffman code dictionary for a source with known probability model
`huffmanenco`	Huffman encoder

Pulse Shaping

The functions in the table below enable you to perform rectangular pulse shaping at a transmitter and matched filtering at the corresponding receiver.

Function	Purpose
`intdump`	Integrate and dump
`rectpulse`	Rectangular pulse shaping

These functions can be useful in conjunction with the modulation functions listed below.

Utility Functions

The toolbox now includes the following utility functions, details of which are on the corresponding reference pages.

Function	Purpose
noisebw	Equivalent noise bandwidth of a filter
qfunc	Q function
qfuncinv	Inverse Q function

Enhancements for Modulation

The functions in the tables below enable you to perform modulation and demodulation using analog and digital methods. Some of the functions support modulation types that Communications Toolbox did not previously support (DPSK and OQPSK). Other functions enhance and replace the older modulation and demodulation functions in Communications Toolbox. The new modulation and demodulation functions are designed to be easier to use than the older ones. Note, however, that the current set of modulation functions supports only analog passband and digital baseband modulation.

Analog Passband Modulation

Function	Purpose
amdemod	Amplitude demodulation
ammod	Amplitude modulation
fmdemod	Frequency demodulation
fmmod	Frequency modulation
pmdemod	Phase demodulation
pmmod	Phase modulation
ssbdemod	Single sideband amplitude demodulation
ssbmod	Single sideband amplitude modulation

Digital Baseband Modulation

Function	Purpose
dpskdemod	Differential phase shift keying demodulation
dpskmod	Differential phase shift keying modulation

Digital Baseband Modulation (Continued)

Function	Purpose
fskdemod	Frequency shift keying demodulation
fskmod	Frequency shift keying modulation
genqamdemod	General quadrature amplitude demodulation
genqammod	General quadrature amplitude modulation
modnorm	Scaling factor for normalizing modulation output
oqpskdemod	Offset quadrature phase shift keying demodulation
oqpskmod	Offset quadrature phase shift keying modulation
pamdemod	Pulse amplitude demodulation
pammod	Pulse amplitude modulation
pskdemod	Phase shift keying demodulation
pskmod	Phase shift keying modulation
qamdemod	Quadrature amplitude demodulation
qammod	Quadrature amplitude modulation

Enhancements for BCH Coding

The functions in the table below enable you to encode and decode BCH codes. These functions enhance and replace the older BCH coding functions in Communications Toolbox.

Function	Purpose
bchdec	BCH decoder
bchenc	BCH encoder
bchgenpoly	Generator polynomial of BCH code

When processing codes using these functions, you can control the primitive polynomial used to describe the Galois field containing the code symbols and the position of the parity symbols.

Updating Existing Modulation MATLAB Code

Compatibility Considerations

If your existing MATLAB code performs modulation or demodulation, then you might want to update it to use the enhanced modulation or demodulation capabilities. Here are some important points to keep in mind:

- The toolbox no longer supports digital passband modulation/demodulation. However, it supports digital baseband modulation/demodulation, which is usually preferable.
- The toolbox no longer supports analog baseband modulation/demodulation. However, it supports analog passband modulation/demodulation.
- The new suite of functions includes a different function for each supported modulation type, whereas the old suite of functions included a smaller number of functions that each supported many modulation types.
- The new modulation/demodulation functions do not apply rectangular pulse shaping when modulating, and do not downsample when demodulating. Also, the new functions' syntax does not involve `Fd`, the sampling rate of the modulator input. To imitate the old functions' behavior, see the new `rectpulse` and `intdump` functions.
- In most cases, the new functions use different kinds of input arguments to describe parameters of the modulation or demodulation scheme. The new sets of arguments are meant to be easier to use, but determining how to update code might not be obvious. To make the task easier, compare the documentation for the old and new functions and compare the functions' outputs for small or well-understood data sets.

Updating Existing BCH MATLAB Code

Compatibility Considerations

If your existing MATLAB code processes BCH codes, then you might want to update it to use the enhanced BCH capabilities. Here are some important points to keep in mind:

- Use `bchenc` instead of `bchenco` and `encode(...,'bch')`.
- Use `bchdec` instead of `bchdeco` and `decode(...,'bch')`.

- Use `bchgenpoly` instead of `bchpoly`.
- `bchenc` and `bchdec` use Galois arrays for the messages and codewords. To learn more about Galois arrays, see "Representing Elements of Galois Fields" in the Communications Toolbox User's Guide.
- `bchenc` places (and `bchdec` expects to find) the parity symbols at the *end* of each word by default. To process codes in which the parity symbols are at the beginning of each word, use the string `'beginning'` as the last input argument when you invoke `bchenc` and `bchdec`.

Converting Between Release 13 and Release 14 Representations of Code Data. To help you update your existing MATLAB code that processes BCH codes, the example below illustrates how to encode data using the new `bchenc` function and the earlier `encode` and `bchenco` functions.

```
% Basic parameters for coding
n = 15; k = 11; % Message length and codeword length
w = 10; % Number of words to encode in this example

% R13 binary vector format
mydata_r13 = randint(w*k,1); % Long vector
% R13 binary matrix format
mydata_r13_mat = reshape(mydata_r13,k,w)'; % One message per row
% R13 decimal format
mydata_r13_dec = bi2de(mydata_r13_mat); % Convert to decimal.

% Equivalent R14 Galois array format
mydata_r14 = fliplr(gf(mydata_r13_mat));

% Encode the data using R13 methods.
code_r13 = encode(mydata_r13,n,k,'bch');
code_r13_mat = encode(mydata_r13_mat,n,k,'bch');
code_r13_dec = encode(mydata_r13_dec,n,k,'bch/decimal');
code_r13_bchenco = bchenco(mydata_r13_mat,n,k);

% Encode the data using R14 method.
code_r14 = bchenc(mydata_r14,n,k);
codeX = fliplr(double(code_r14.x)); % Retrieve from Galois array.

% Check that all resulting codes are the same.
```

```
% c1, c2, c3, and c4 should all be true.
c1 = isequal(de2bi(code_r13_dec),code_r13_mat);
c2 = isequal(reshape(code_r13,n,w)',code_r13_mat);
c3 = isequal(code_r13_bchenco,code_r13_mat);
c4 = isequal(code_r13_mat,codeX); % Compare R13 with R14.
```

Changes in Functionality

Compatibility Considerations

The `encode` and `decode` functions no longer perform BCH encoding and decoding. Use the `bchenc` and `bchdec` functions instead.

Obsolete Functions

Compatibility Considerations

The table below lists functions that are obsolete. Although they are included in Release 13 for backward compatibility, they might be removed in a future release. The second column lists functions that provide similar functionality. In some cases, the similar function requires different input arguments or produces different output arguments, compared to the original function.

Obsolete Function	Similar Function in R14
`ademod`	`amdemod`, `fmdemod`, `pmdemod`, `ssbdemod`
`ademodce`	Use passband demodulation instead: `amdemod`, `fmdemod`, `pmdemod`, `ssbdemod`
`amod`	`ammod`, `fmmod`, `pmmod`, `ssbmod`
`amodce`	Use passband modulation instead: `ammod`, `fmmod`, `pmmod`, `ssbmod`
`apkconst`	`genqammod` or `pskmod` for mapping; `scatterplot` for plotting
`bchdeco`	`bchdec`
`bchenco`	`bchenc`

Obsolete Function	Similar Function in R14
`bchpoly`	`bchgenpoly`
`ddemod`	Use baseband demodulation instead: `genqamdemod`, `pamdemod`, `pskdemod`, `qamdemod`, `fskdemod`
`ddemodce`	`genqamdemod`, `pamdemod`, `pskdemod`, `qamdemod`, `fskdemod`
`demodmap`	`genqamdemod`, `pamdemod`, `pskdemod`, `qamdemod`
`dmod`	Use baseband modulation instead: `genqammod`, `pammod`, `pskmod`, `qammod`, `fskmod`
`dmodce`	`genqammod`, `pammod`, `pskmod`, `qammod`, `fskmod`
`modmap`	`genqammod`, `pammod`, `pskmod`, `qammod` for mapping; `scatterplot` for plotting
`qaskdeco`	`qamdemod`
`qaskenco`	`qammod` for mapping; `scatterplot` for plotting

Version 2.1 (R13) Communications Toolbox Software

This table summarizes what's new in Version 2.1 (R13):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Fixed bugs and known problems	No

New features and changes introduced in this version are

Galois Field Computations

Communications Toolbox supports a new data type that allows you to manipulate arrays of elements of a Galois field having 2^m elements, where m is an integer between 1 and 16. When you use this data type, most computations have the same syntax that you would use to manipulate ordinary MATLAB arrays of real numbers. The consistency with MATLAB syntax makes the new Galois field capabilities easier to use than the analogous Release 12 capabilities.

Enhancements for Reed-Solomon Codes

The functions in the table below allow you to encode and decode Reed-Solomon codes, including shortened Reed-Solomon codes. These functions enhance and replace the older Reed-Solomon coding functions in Communications Toolbox.

Function	Purpose
`rsdec`	Reed-Solomon decoder
`rsenc`	Reed-Solomon encoder
`rsgenpoly`	Generator polynomial of Reed-Solomon code

When processing codes using these functions, you can control the generator polynomial, the primitive polynomial used to describe the Galois field containing the code symbols, and the position of the parity symbols.

Arithmetic Coding

The functions in the table below allow you to perform arithmetic coding.

Function	Purpose
`arithdeco`	Decode binary code using arithmetic decoding
`arithenco`	Encode a sequence of symbols using arithmetic coding

Fixed Bugs

Reed-Solomon decoder corrects up to t errors

The new function `rsdec` accurately decodes Reed-Solomon codes containing up to `t` errors in each codeword. This new function replaces the earlier functions `rsdeco` and `rsdecode`.

Reed-Solomon encoder and decoder use more conventional format for data

The new functions `rsenc` and `rsdec` operate on the new Galois data type, which represents symbols using a decimal format.

The new functions enable you to choose whether parity bits appear at the beginning or end of each codeword.

Known Problems

Galois field manipulations cannot be compiled

The Galois field data type is not compatible with the MATLAB Compiler.

Incorrect name of data file in printed documentation

The section "Speed and Nondefault Primitive Polynomials" in the printed manual refers to a MAT-file called `gftable.mat`. It should say `userGftable.mat`.

Updating Existing Galois Field Code

Compatibility Considerations

If your existing code performs computations in Galois fields having 2^m elements, where m is an integer between 1 and 16, then you might want to update your code to use the new Galois field capabilities.

Replacing Functions. The table below lists Release 12 functions that correspond to Release 13 functions or operators acting on the new Galois field data type. Compared to the syntax of their Release 12 counterparts, the syntaxes of the Release 13 functions are different, but generally easier to use.

Release 12 Function	Release 13 Function or Operator	Comments
`gfadd`	`+`	
`gfconv`	`conv`	

Release 12 Function	Release 13 Function or Operator	Comments
`gfcosets`	`cosets`	`cosets` returns a cell array, whereas `gfcosets` returns a NaN-padded matrix.
`gfdeconv`	`deconv`	
`gfdiv`	`./`	
`gffilter`	`filter`	Unlike `gffilter`, `filter` also returns the final states.
`gflineq`	`\`	
`gfplus`	`+`	
`gfprimck`	`isprimitive`	`isprimitive` detects primitivity but not reducibility.
`gfprimdf`	`primpoly`	
`gfprimfd`	`primpoly`	
`gfrank`	`rank`	
`gfroots`	`roots`	Unlike `gfroots`, `roots` indicates multiplicities of roots and can process polynomials in an extension field
`gfsub`	`-`	
`gftuple`	`.^`, `log`, `polyval`	See "Converting and Simplifying Formats Using R13 Galois Arrays" on page 53 for more details.

Converting Between Release 12 and Release 13 Representations of Field Elements. In some parts of your existing code, you might need to convert data between the exponential format supported in Release 12 and the new Galois array. The code example below performs such conversions on a sample vector that represents elements of GF(16).

```
% Sample data
m = 4; % For example, work in GF(2^4) = GF(16).
a_r12 = [2 5 0 -Inf]; % GF(16) elements in exponential format

% 1. Convert to the Release 13 Galois array.
A = gf(2,m); % Primitive element of the field
a_r13 = A.^(a_r12); % Positive exponents mean A to that power.
a_r13(find(a_r12 < 0)) = 0; % Negative exponents mean zero.

% 2. Convert back to the Release 12 exponential format.
m = a_r13.m; A = gf(2,m);
a_r12again = zeros(size(a_r13)); % Preallocate space in a matrix.
zerolocations = find(a_r13 == 0);
nonzerolocations = find(a_r13 ~= 0);
a_r12again(zerolocations) = -Inf; % Map 0 to negative exponent.
a_r12again(nonzerolocations) = log(a_r13(nonzerolocations));

% Check that the two conversions are inverses.
ck = isequal(a_r12,a_r12again)

ck =

     1
```

Converting Between Release 12 and Release 13 Representations of Polynomials. Release 12 and Release 13 use different formats for representing polynomials over GF(2^m). Release 12 represents a polynomial as a vector of coefficients in order of *ascending* powers. Depending on the context, each coefficient listed in the vector represents either an element in a prime field or the exponential format of an element in an extension field. Release 13 uses the conventions described below.

Primitive Polynomials

The functions `gf`, `isprimitive`, and `primpoly` represent a primitive polynomial using an integer scalar whose binary representation lists the coefficients of the polynomial. The least significant bit is the constant term.

For example, the scalar 13 has binary representation 1101 and represents the polynomial $D^3 + D^2 + 1$.

Other Polynomials

When performing arithmetic with, evaluating, or finding roots of a polynomial, or when finding a minimal polynomial of a field element, you represent the polynomial using a Galois vector of coefficients in order of *descending* powers. Each coefficient listed in the vector represents an element in the field using the representation described in "How Integers Correspond to Galois Field Elements".

For example, the Galois vector `gf([1 1 0 1],1)` represents the polynomial $x^3 + x^2 + 1$. Also, the Galois vector `gf([1 2 3],3)` represents the polynomial $x^2 + Ax + (A+1)$, where A is a root of the default primitive polynomial for $GF(2^3)$. The coefficient of A+1 corresponds to the vector entry of 3 because the binary representation of 3 is 11.

Example Showing Conversions

The code example below might help you determine how to convert between the Release 12 and Release 13 formats for polynomials.

```
m = 3; % Work in GF(8).

poly_r12 = [1 1 0 1]; % 1+x+x^3, ascending order
poly_r13 = gf([1 0 1 1],m); % x^3+x+1 in GF(8), descending order

% R12 polynomials
pp_r12 = gfprimdf(m); % A primitive polynomial
mp_r12 = gfminpol(4,m); % The minimal polynomial of an element
rts_r12 = gfroots(poly_r12); % Find roots.
```

```
% R13 polynomials
pp_r13 = primpoly(m,'nodisplay'); % A primitive polynomial
mp_r13 = minpol(gf(4,m)); % The minimal polynomial of an element
rts_r13 = roots(poly_r13); % Find roots.

% R12 polynomials converted to R13 formats
% For primitive poly, change binary vector to decimal scalar.
pp_r12_conv = bi2de(pp_r12);
% For minimal poly, change ordering and make it a Galois array.
mp_r12_conv = gf(fliplr(mp_r12));
% For roots of polynomial, note that R12 answers are in
% exponential format. Convert to Galois array format.
rts_r12_conv = gf(2,m) .^ rts_r12;

% Check that R12 and R13 yield the same answers.
c1 = isequal(pp_r13,pp_r12_conv); % True.
c2 = isequal(mp_r13,mp_r12_conv); % True.
c3 = isequal(rts_r13,rts_r12_conv); % True.
```

Converting and Simplifying Formats Using R13 Galois Arrays. If your existing code uses `gftuple` to convert between exponential and polynomial formats, or to simplify one of these formats, then the code example below might help you determine how to perform those tasks using the Release 13 Galois array.

```
% First define key characteristics of the field.
m = 4; % For example, work in GF(2^4) = GF(16).
A = gf(2,m); % Primitive element of the field

% 1. Simplifying a Polynomial Format
poly_big = 2^10 + 2^7;
% Want to refer to the element A^10 + A^7. However,
% cannot use gf(poly_big,m) because poly_big is too large.
poly1 = A.^10 + A.^7 % One way to define the element.
poly2 = polyval(de2bi(poly_big,'left-msb'),A); % Another way.
% The results show that A^10 + A^7 equals A^3 + A^2 in this
% field, using the binary representation of 12 as 1100.

% 2. Simplifying an Exponential Format
```

```
exp_big = 39;
exp_simple = log(A.^exp_big) % Simplest exponential format.
% The results show that A^39 equals A^9 in this field.

% 3. Converting from Exponential to Polynomial Format
expf1 = 7;
pf1 = A.^expf1
% The results show that A^7 equals A^3 + A + 1 in this
% field, using the binary representation of 11 as 1011.

% 4. Converting from Polynomial to Exponential Format
pf2 = 11; % Represents the element A^3 + A + 1
expf2 = log(gf(pf2,m))
% The results show that A^3 + A + 1 equals A^7 in this field.
```

The output is below.

```
poly1 = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

Array elements =

    12


exp_simple =

     9


pf1 = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

Array elements =

    11


expf2 =

     7
```

Updating Existing Reed-Solomon MATLAB Code

Compatibility Considerations

If your existing MATLAB code processes Reed-Solomon codes, then you might want to update it to use the enhanced Reed-Solomon capabilities. Below are some important points to keep in mind:

- Use `rsenc` instead of `rsenco`, `rsencode`, and `encode(...,'rs')`.
- Use `rsdec` instead of `rsdeco`, `rsdecode`, and `decode(...,'rs')`.
- Use `rsgenpoly` instead of `rspoly`.
- `rsenc` and `rsdec` use Galois arrays for the messages and codewords. To learn more about Galois arrays, see "Representing Elements of Galois Fields".
- `rsenc` and `rsdec` interpret symbols in a different way compared to the Release 12 functions. For an example showing how to convert between Release 12 and Release 13 interpretations, see "Converting Between Release 12 and Release 13 Representations of Code Data" on page 56.
- The Release 12 functions support three different data formats. The exponential format is most easily converted to the Release 13 format. To convert your data among the various Release 12 formats as you prepare to upgrade to the new Release 13 functions, see "Converting Among Various Release 12 Representations of Coding Data" on page 57.
- `rsenc`, `rsdec`, and `rsgenpoly` use a Galois array in *descending* order to represent the generator polynomial argument. The commands below indicate how to convert generator polynomials from the Release 12 format to the Release 13 format.

```
n = 7; k = 3; % Examples of code parameters
m = log2(n+1); % Number of bits in each symbol
gp_r12 = rspoly(n,k); % R12 exponential format, ascending order
gp_r13 = gf(2,m).^fliplr(gp_r12); % Convert to R13 format.
```

- `rsenc` places (and `rsdec` expects to find) the parity symbols at the *end* of each word by default. To process codes in which the parity symbols are at the beginning of each word, use the string `'beginning'` as the last input argument when you invoke `rsenc` and `rsdec`.

Converting Between Release 12 and Release 13 Representations of Code Data. To help you update your existing M-code that processes Reed-Solomon codes, the example below illustrates how to encode data using the new `rsenc` function and the earlier `rsenco` function.

```
% Basic parameters for coding
m = 4; % Number of bits per symbol in each codeword
t = 2; % Error-correction capability
n = 2^m-1; k = n-2*t; % Message length and codeword length
w = 10; % Number of words to encode in this example

% Lookup tables to translate formats between rsenco and rsenc
p2i = [0 gf(2,m).^[0:2^m-2]]; % Galois vector listing powers
i2p = [-1 log(gf(1:2^m-1,m))]; % Integer vector listing logs

% R12 method, exponential format
% Exponential format uses integers between -1 and 2^m-2.
mydata_r12 = randint(w,k,2^m)-1;
code_r12 = rsenco(mydata_r12,n,k,'power'); % * Encode the data. *
% Convert any -Inf values to -1 to facilitate comparisons.
code_r12(isinf(code_r12)) = -1;
code_r12 = reshape(code_r12,n,w)'; % One codeword per row

% R12 method, decimal format
% This yields same results as R12 exponential format.
mydata_r12_dec = mydata_r12 + 1; % Convert to decimal.
code_r12_dec = rsenco(mydata_r12_dec,n,k,'decimal'); % Encode.
code_r12_dectoexp = code_r12_dec - 1; % Convert to exponential.
c1 = isequal(code_r12,code_r12_dectoexp); % True.

% R12 method, binary format
% This yields same results as R12 exponential format.
mydata_r12_bin = de2bi(mydata_r12_dec',m); % Convert to binary.
code_r12_bin = rsenco(mydata_r12_bin,n,k,'binary'); % Encode.
code_r12_bintoexp = reshape(bi2de(code_r12_bin),n,w)' - 1;
c2 = isequal(code_r12,code_r12_bintoexp); % True.

% R13 method
mydata_r13 = fliplr(mydata_r12); % Reverse the order.
% Convert format, using +2 to get in the right range for indexing.
```

```
mydata_r13 = p2i(mydata_r13+2);
code_r13 = rsenc(mydata_r13,n,k); % * Encode the data. *
codeX = double(code_r13.x); % Retrieve data from Galois array.
% Convert format, using +1 to get in the right range for indexing.
codelogX = i2p(codeX+1);
codelogX = fliplr(codelogX); % Reverse the order again.

c3 = isequal(code_r12,codelogX) % True.

c3 =

     1
```

Converting Among Various Release 12 Representations of Coding Data. These rules indicate how to convert among the exponential, decimal, and binary formats that the Release 12 Reed-Solomon functions support:

- To convert from decimal format to exponential format, subtract one.
- To convert from exponential format to decimal format, replace any negative values by -1 and then add one.
- To convert between decimal and binary formats, use `de2bi` and `bi2de`. The right-most bit is the most significant bit in this context.

The commands below illustrate these conversions.

```
msgbin = randint(11,4); % Message for a (15,11) = (2^4-1, 11) code
msgdec = bi2de(msgbin)'; % Binary to decimal
msgexp = msgdec - 1; % Decimal to exponential
codeexp = rsenco(msgexp,15,11,'power');
codeexp(find(codeexp < 0)) = -1; % Use -1 consistently.
codedec = codeexp + 1; % Exponential to decimal
codebin = de2bi(codedec); % Decimal to binary
```

Changes in Functionality

Compatibility Considerations

The table below lists functions whose behavior has changed.

Function	Change in Functionality
`wgn`	The default measurement unit is the dBW, formerly documented as "dB." To specify this unit explicitly in the syntax, set the *`powertype`* input argument to `'dBW'`, not `'dB'`. The output of the function is unaffected by this change in syntax.

Obsolete Functions

Compatibility Considerations

The table below lists functions that are obsolete. Although they are included in Release 13 for backward compatibility, they might be removed in a future release. The second column lists functions that provide similar functionality. In some cases, the similar function requires different input arguments or produces different output arguments, compared to the original function.

Function	Similar Function
`gfplus`	+ operator for Galois arrays
`rsdeco`	`rsdec`
`rsdecode`	`rsdec`
`rsenco`	`rsenc`
`rsencode`	`rsenc`
`rspoly`	`rsgenpoly`

Version 2.0 (R12) Communications Toolbox Software

This table summarizes what's new in Version 2.0 (R12):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Fixed bugs	No

New features and changes introduced in this version are

Convolutional Coding Functions

Communications Toolbox processes feedforward and feedback convolutional codes that can be described by a trellis structure or a set of generator polynomials. It uses the Viterbi algorithm to implement hard-decision and soft-decision decoding. These new functions support convolutional coding:

- `convenc` creates a convolutional code from binary data.
- `vitdec` decodes convolutionally encoded data using the Viterbi algorithm.
- `poly2trellis` converts a polynomial description of a convolutional encoder to a trellis description.

- `istrellis` checks if the input is a valid trellis structure representing a convolutional encoder.

Gaussian Noise Functions

These new functions create Gaussian noise:

- `awgn` adds white Gaussian noise to the input signal to produce a specified signal-to-noise ratio.
- `wgn` generates white Gaussian noise with a specified power, impedance, and complexity.

Other New Functions

These functions are also new in Release 12:

- `eyediagram` plots an eye diagram.
- `marcumq` implements the generalized Marcum Q function.
- `oct2dec` converts octal numbers to decimal numbers.
- `randerr` generates bit error patterns. This is similar to the obsolete function `randbit`, but it accepts a more intuitive set of input arguments and uses an upgraded random number generator.
- `randsrc` generates random matrices using a prescribed alphabet.
- `scatterplot` produces a scatter plot.
- `syndtable` generates syndrome decoding tables. This is similar to the obsolete function `htruthtb`, but it is not limited to single-error-correction codes.

Enhancements to Existing Functions

The following functions have been enhanced in Release 12:

- `biterr` and `symerr` provide a third output argument that indicates the results of individual comparisons. These functions also provide more comprehensive support for comparisons between a vector and a matrix.

- `de2bi` and `bi2de` use an optional input flag to indicate the ordering of bits. If you omit the flag from the list of input arguments, then the default behavior matches that of Release 11.
- `randint` can operate without input arguments. Also, it can accept a negative value for the optional third input argument.

Fixed Bugs

VITERBI is slow and does not decode correctly

VITERBI has been replaced by a new function, VITDEC, which is much faster and decodes correctly.

DDEMOD and DDEMODCE do not produce correct symbol error rates

DDEMOD and DDEMODCE now produce the optimal symbol error rate in AWGN for PSK, ASK, QASK (QAM), FSK, and noncoherent FSK.

DMOD and DMODCE generate incorrect waveform for MSK and FSK

DMOD and DMODCE now generate the correct waveform for MSK and FSK.

GFADD, GFSUB, GFDIV, GFMUL, GFCONV and GFDECONV return incorrect answers

GFADD, GFSUB, GFDIV, GFMUL, GFCONV and GFDECONV have been improved in the following ways:

- Correct answers for prime and extension Galois fields, including prime fields, GF(p), where p>2.
- Correct handling of -Inf and negative values for extension Galois fields.
- Enhanced help descriptions to better distinguish the purposes of the functions.
- Improved input parameter checking.

GFMINPOL returns incorrect answers when first input is -Inf or when p>2. The function also sometimes crashes

GFMINPOL now returns the correct answers and does not crash.

GFPLUS and RSENCODE returns incorrect answers for negative inputs

GFPLUS now returns correct answers for negative inputs. This fix also allows RSENCODE to return correct results.

GFLINEQ returns incorrect answers in prime Galois fields of order greaterthan 2

GFLINEQ now solves linear equations in prime Galois fields of order greater than 2.

GFPRIMDF produces "out of memory" messages for degrees higher than 24

GFPRIMDF now uses less memory and can find primitives of degrees greater than 24. However, this calculation will take considerable time.

DECODE using the cyclic decoder option does not decode (23,12) Golay codecorrectly

The cyclic decoder now decodes the (23,12) Golay code correctly.

GFPRIMFD finds incorrect primitive polynomial

GFPRIMFD finds the correct primitive polynomial for the given Galois field.

GFTUPLE returns incorrect answers when m=1

GFTUPLE now returns the correct answers when m=1.

GFPRIMCK, GFTRUNC, GFADD and GFFILTER causes segmentation violations

GFPRIMCK, GFTRUNC, GFADD and GFFILTER do not cause segmentation violations.

GFPRIMCK returns incorrect answers if p > 2 or inputs are large.

GFPRIMCK correctly determines if a polynomial is irreducible and/or primitive.

DECODE incorrectly decodes block codes

DECODE now correctly decodes block codes using either the [I P] or [P I] standard forms of the generator and parity-check matrices.

RCOSFLT does not correctly filter and upsample the input signal

RCOSFLT now applies the correct raised-cosine filter type and fully filters and upsamples the input signal.

EYESCAT is difficult to use and plots I and Q components together

EYESCAT has been replaced by new functions, EYEDIAGRAM and SCATTERPLOT, which are easier to use, plot I and Q components separately, and allow X-Y plots.

ADEMOD ignores the phase offset parameter under the 'pm' option andhas no sensitivity parameter under the 'fm' option

The phase offset now causes the correct phase offset in the demodulator. New parameters were introduced to allow the sensitivity to be changed.

ADEMOD 'pm' option - sensitivity parameter is required and causes a dcoffset

ADEMOD now has an optional parameter called 'VCOconst' that replaces sensitivity and does not cause a dc offset.

RANDINT hangs when the range is large

RANDINT no longer hangs for large numbers.

RANDBIT output is not random

RANDBIT has been replaced by a new function, RANDERR, which generates random output and supports for seeding.

Changes in Functionality

Compatibility Considerations

The table below lists functions whose behavior has changed.

Function	Change in Functionality
`bi2de`	Distinguishes between rows and columns as input vectors. Treats column vector as separate numbers, not as digits of a single number. To adapt your existing code, transpose the input vector if necessary.
`biterr`	Input argument `k` must be large enough to represent all elements of the input arguments `x` and `y`.
`biterr`, `symerr`	Distinguish between rows and columns as input vectors. To adapt your existing code, transpose the input vector if necessary.
	Use different strings for the input argument that controls row-wise and column-wise comparisons.
	Produce vector, not scalar, output if one input is a vector. See these functions' reference pages for more information.

Function	Change in Functionality
`de2bi`	Second input argument, if it appears, must not be smaller than the number of bits in any element of the first input argument. Previously, the function produced a truncated binary representation instead of an error. To adapt your existing code, specify a sufficiently large number for the second input argument and then truncate the answer manually.
`ddemod`	Default behavior uses no filter, not a Butterworth filter. Regardless of filtering, the function uses an integrator to perform demodulation.
`dmod`, `ddemod`, `dmodce`, `ddemodce`, `modmap`, `demodmap`	For frequency shift keying method, the default separation between successive frequencies is `Fd`, not `2*Fd/M`. For minimum shift keying method, the separation between frequencies is `Fd/2`, not `Fd`.
`encode`, `decode`	No longer support convolutional coding. Use `convenc` and `vitdec` instead.
`gflineq`	If the equation has no solutions, then the function returns an empty matrix, not a matrix of zeros.
`randint`	Uses state instead of seed to initialize random number generator. See `rand` for more information about initializing random number generators.
`rcosflt`	The `'wdelay'` flag is superfluous. The function now behaves as the Release 11 function behaved with the `'wdelay'` flag.

Obsolete Functions

Compatibility Considerations

The table below lists functions that are obsolete. Although they are included in Release 12 for backward compatibility, they might be removed in a future release. Where applicable, the second column lists functions that provide similar functionality. In some cases, the similar function requires different arguments or produces different results compared to the original function.

Function	Similar Function, if Any
`commgui`	
`convdeco`	`vitdec`
`convenco`	`convenc`
`eyescat`	`eyediagram`, `scatterplot`
`flxor`	`bitxor`
`gen2abcd`	
`htruthtb`	`syndtable`
`imp2sys`	
`oct2gen`	
`randbit`	`randerr`
`sim2gen`	
`sim2logi`	
`sim2tran`	
`viterbi`	`vitdec`

Communications Toolbox Release Notes Compatibility Summary

This table summarizes new features and changes that might cause incompatibilities when you upgrade from an earlier version, or when you use files on multiple versions. Details are provided in the description of the new feature or change.

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version V4.5 (R2010b)	None
V4.5 (R2010a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Error Rate Test Console Enhancements" on page 6
V4.4 (R2009b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Channel Objects Support Parallel Computing Toolbox" on page 11 • "Functions and Function Elements Being Removed" on page 13
V4.3 (R2009a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • commsrc.package now supports PN sequence generation
V4.2 (R2008b)	None
V4.1 (R2008a)	None
V4.0 (R2007b)	None

Version (Release)	New Features and Changes with Version Compatibility Impact
V3.5 (R2007a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Theoretical Results Refined for `berawgn`, `berfading`, and BERTool" on page 27
V3.4 (R2006b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Object-Based PSK and QAM Modulation and Demodulation" on page 30 • "QAM and PSK Modulation and Demodulation Functions Obsoleted" on page 30
V3.3 (R2006a)	None
V3.2 (R14SP3)	None
V3.1 (R14SP2)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "gfrank" on page 34 • "encode, decode, and quantiz" on page 34
V3.0.1 (R14SP1)	None

Version (Release)	New Features and Changes with Version Compatibility Impact
V3.0 (R14)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Updating Existing Modulation MATLAB Code" on page 43 • "Updating Existing BCH MATLAB Code" on page 43 • "Changes in Functionality" on page 45 • "Obsolete Functions" on page 45
V2.1 (R13)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Updating Existing Galois Field Code" on page 49 • "Updating Existing Reed-Solomon MATLAB Code" on page 55 • "Changes in Functionality" on page 57 • "Obsolete Functions" on page 58
V2.0 (R12)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Changes in Functionality" on page 64 • "Obsolete Functions" on page 66