

Communications Toolbox Release Notes

Summary by Version	1
About Release Notes	1
Version 3.3 (R2006a) Communications Toolbox	4
convenc and vitdec Updated with Puncturing and Erasing	4
Enhanced pamdemod, pskdemod, and qamdemod Functions	4
Version 3.2 (R14SP3) Communications Toolbox	5
Added function bchnumerr	5
Speed increase in bchgenpoly, bchenc, and bchdec functions	5
Version 3.1 (R14SP2) Communications Toolbox	6
Channel Visualization Tool	6
Improved Rayleigh Fading Channel	6
Gray Coding Functionality	6
Rician Channel Enhancement to the BERTool	6
gfrank	7
encode, decode, and quantiz	7
Version 3.0.1 (R14SP1) Communications Toolbox	8
Rician Channel BER Calculations	8
berfading Updated for Rician Channel	8
New Adaptive Equalization Demo	8
Version 3.0 (R14) Communications Toolbox	9
Bit Error Rate Analysis GUI	10
Performance Evaluation	10
Equalizers	10
Fading Channels and Binary Symmetric Channel	11
Interleavers	12
Huffman Coding	13
Pulse Shaping	13
Utility Functions	13
Enhancements for Modulation	14

Enhancements for BCH Coding	15
Updating Existing Modulation M-Code	16
Updating Existing BCH M-Code	17
Changes in Functionality	18
Obsolete Functions	18
Version 2.1 (R13) Communications Toolbox	20
Galois Field Computations	20
Enhancements for Reed-Solomon Codes	21
Arithmetic Coding	21
Fixed Bugs	22
Known Problems	22
Updating Existing Galois Field Code	22
Updating Existing Reed-Solomon M-Code	28
Changes in Functionality	30
Obsolete Functions	31
Version 2.0 (R12) Communications Toolbox	32
Convolutional Coding Functions	32
Gaussian Noise Functions	33
Other New Functions	33
Enhancements to Existing Functions	33
Fixed Bugs	34
Changes in Functionality	37
Obsolete Functions	39
Compatibility Summary for Communications Toolbox	40

Summary by Version

This table provides quick access to what's new in each version. For clarification, see “Summary by Version” on page 1.

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V3.3 (R2006a)	Yes Details	No	Bug Reports at Web site	Printable Release Notes: PDF V3.3 product documentation
V3.2 (R14SP3)	Yes Details	No	Bug Reports at Web site	No
V3.1 (R14SP2)	Yes Details	Yes Summary	Bug Reports at Web site	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

About Release Notes

Use release notes when upgrading to a newer version to learn about new features and changes, and the potential impact on your existing files and practices. Release notes are also beneficial if you use or support multiple versions.

If you are not upgrading from the most recent previous version, review release notes for all interim versions, not just for the version you are installing. For example, when upgrading from V1.0 to V1.2, review the New Features and

Changes, Version Compatibility Considerations, and Bug Reports for V1.1 and V1.2.

New Features and Changes

These include

- New functionality
- Changes to existing functionality
- Changes to system requirements (complete system requirements for the current version are at the MathWorks Web site)
- Any version compatibility considerations associated with each new feature or change

Version Compatibility Considerations

When a new feature or change introduces a known incompatibility between versions, its description includes a **Compatibility Considerations** subsection that details the impact. For a list of all new features and changes that have compatibility impact, see the “Compatibility Summary for Communications Toolbox” on page 40.

Compatibility issues that become known after the product has been released are added to Bug Reports at the MathWorks Web site. Because bug fixes can sometimes result in incompatibilities, also review fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

MathWorks Bug Reports is a user-searchable database of known problems, workarounds, and fixes. The MathWorks updates the Bug Reports database as new problems and resolutions become known, so check it as needed for the latest information.

Access Bug Reports at the MathWorks Web site using your MathWorks Account. If you are not logged in to your MathWorks Account when you link to Bug Reports, you are prompted to log in or create an account. You then can view bug fixes and known problems for R14SP2 and more recent releases.

Related Documentation at Web Site

Printable Release Notes (PDF). You can print release notes from the PDF version, located at the MathWorks Web site. The PDF version does not support links to other documents or to the Web site, such as to Bug Reports. Use the browser-based version of release notes for access to all information.

Product Documentation. At the MathWorks Web site, you can access complete product documentation for the current version and some previous versions, as noted in the summary table.

Version 3.3 (R2006a) Communications Toolbox

This table summarizes what's new in V3.3 (R2006a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports at Web site	Printable Release Notes: PDF V3.3 product documentation

New features and changes introduced in this version are described here:

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

This table summarizes what's new in V3.2 (R14SP3):

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

New features and changes introduced in this version are described here:

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 (R14SP2) Communications Toolbox

This table summarizes what's new in V3.1 (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 at Web site	No

New features and changes introduced in this version are described here:

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

This table summarizes what's new in V3.0.1 (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 described here:

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

This table summarizes what's new in V3.0 (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” on page 10
- “Performance Evaluation” on page 10
- “Equalizers” on page 10
- “Fading Channels and Binary Symmetric Channel” on page 11
- “Interleavers” on page 12
- “Huffman Coding” on page 13
- “Pulse Shaping” on page 13
- “Utility Functions” on page 13
- “Enhancements for Modulation” on page 14
- “Enhancements for BCH Coding” on page 15
- “Updating Existing Modulation M-Code” on page 16
- “Updating Existing BCH M-Code” on page 17
- “Changes in Functionality” on page 18
- “Obsolete Functions” on page 18

Bit Error Rate Analysis GUI

The 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
dfc	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
mlseq	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
<code>algdeintrlv</code>	Restore ordering of symbols, using algebraically derived permutation table
<code>algintrlv</code>	Reorder symbols, using algebraically derived permutation table
<code>deintrlv</code>	Restore ordering of symbols
<code>helscandintrlv</code>	Restore ordering of symbols in a helical pattern
<code>helscanintrlv</code>	Reorder symbols in a helical pattern
<code>intrlv</code>	Reorder sequence of symbols
<code>matdeintrlv</code>	Restore ordering of symbols by filling a matrix by columns and emptying it by rows
<code>matintrlv</code>	Reorder symbols by filling a matrix by rows and emptying it by columns
<code>randdeintrlv</code>	Restore ordering of symbols, using a random permutation
<code>randintrlv</code>	Reorder symbols, using a random permutation

Convolutional Interleaving

Function	Purpose
<code>convdeintrlv</code>	Restore ordering of symbols, using shift registers
<code>convintrlv</code>	Permute symbols, using shift registers
<code>heldeintrlv</code>	Restore ordering of symbols permuted using <code>helintrlv</code>
<code>helintrlv</code>	Permute symbols, using a helical array

Convolutional Interleaving (Continued)

Function	Purpose
<code>muxdeintrlv</code>	Restore ordering of symbols, using specified shift registers
<code>muxintrlv</code>	Permute symbols, using shift registers with specified delays

Huffman Coding

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

Function	Purpose
<code>huffmandeco</code>	Huffman decoder
<code>huffmandict</code>	Generate Huffman code dictionary for a source with known probability model
<code>huffmanenco</code>	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
<code>intdump</code>	Integrate and dump
<code>rectpulse</code>	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 the Communications Toolbox did not previously support (DPSK and OQPSK). Other functions enhance and replace the older modulation and demodulation functions in the 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
amdmod	Amplitude demodulation
ammod	Amplitude modulation
fmdemod	Frequency demodulation
fmmod	Frequency modulation
pmdemod	Phase demodulation
pmmod	Phase modulation

Analog Passband Modulation (Continued)

Function	Purpose
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
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 the 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 M-Code

Compatibility Considerations

If your existing M-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 F_d , 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 M-Code

Compatibility Considerations

If your existing M-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 M-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
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

This table summarizes what's new in V2.1 (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 described here:

Galois Field Computations (p. 20)

Enhancements for Reed-Solomon Codes (p. 21)

Arithmetic Coding (p. 21)

Fixed Bugs (p. 22)

Known Problems (p. 22)

Updating Existing Galois Field Code (p. 22)

Updating Existing Reed-Solomon M-Code (p. 28)

Changes in Functionality (p. 30)

Obsolete Functions (p. 31)

Galois Field Computations

The 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 the 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	
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 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 M-Code

Compatibility Considerations

If your existing M-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 .
- 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 .
- `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 <i>powertype</i> 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

This table summarizes what's new in V2.0 (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 described here:

Convolutional Coding Functions
(p. 32)

Gaussian Noise Functions (p. 33)

Other New Functions (p. 33)

Enhancements to Existing Functions
(p. 33)

Fixed Bugs (p. 34)

Changes in Functionality (p. 37)

Obsolete Functions (p. 39)

Convolutional Coding Functions

The 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 `htruthb`, 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.

DDEMODO and DDEMODOCE do not produce correct symbol error rates

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

DMODO and DMODOCE generate incorrect waveform for MSK and FSK

DMODO and DMODOCE 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 greater than 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 code correctly

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.

ADEMODO ignores the phase offset parameter under the 'pm' option and has 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.

ADEMODO 'pm' option - sensitivity parameter is required and causes a dc offset

ADEMODO 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, dmodc, ddemodc, modmap, demodmap	For frequency shift keying method, the default separation between successive frequencies is F_d , not $2 \cdot F_d / M$. For minimum shift keying method, the separation between frequencies is $F_d / 2$, not F_d .
encode, decode	No longer support convolutional coding. Use <code>convenc</code> and <code>vitdec</code> instead.
gflinseq	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 <code>rand</code> 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

Compatibility Summary for Communications Toolbox

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 V3.3 (R2006a)	None
V3.2 (R14SP3)	None
V3.1 (R14SP2)	See the Compatibility Considerations subheading for each of these new features or changes: subheading for this new feature or change: <ul style="list-style-type: none"> • “gfrank” on page 7 • “encode, decode, and quantiz” on page 7
V3.0.1 (R14SP1)	None
V3.0 (R14)	See the Compatibility Considerations subheading for each of these new features or changes: subheading for this new feature or change: <ul style="list-style-type: none"> • “Updating Existing Modulation M-Code” on page 16 • “Updating Existing BCH M-Code” on page 17 • “Changes in Functionality” on page 18

Version (Release)	New Features and Changes with Version Compatibility Impact
	<ul style="list-style-type: none"> • “Obsolete Functions” on page 18
<p>V2.1 (R13)</p>	<p>See the Compatibility Considerations subheading for each of these new features or changes: subheading for this new feature or change:</p> <ul style="list-style-type: none"> • “Updating Existing Galois Field Code” on page 22 • “Updating Existing Reed-Solomon M-Code” on page 28 • “Changes in Functionality” on page 30 • “Obsolete Functions” on page 31
<p>V2.0 (R12)</p>	<p>See the Compatibility Considerations subheading for each of these new features or changes: subheading for this new feature or change:</p> <ul style="list-style-type: none"> • “Changes in Functionality” on page 37 • “Obsolete Functions” on page 39