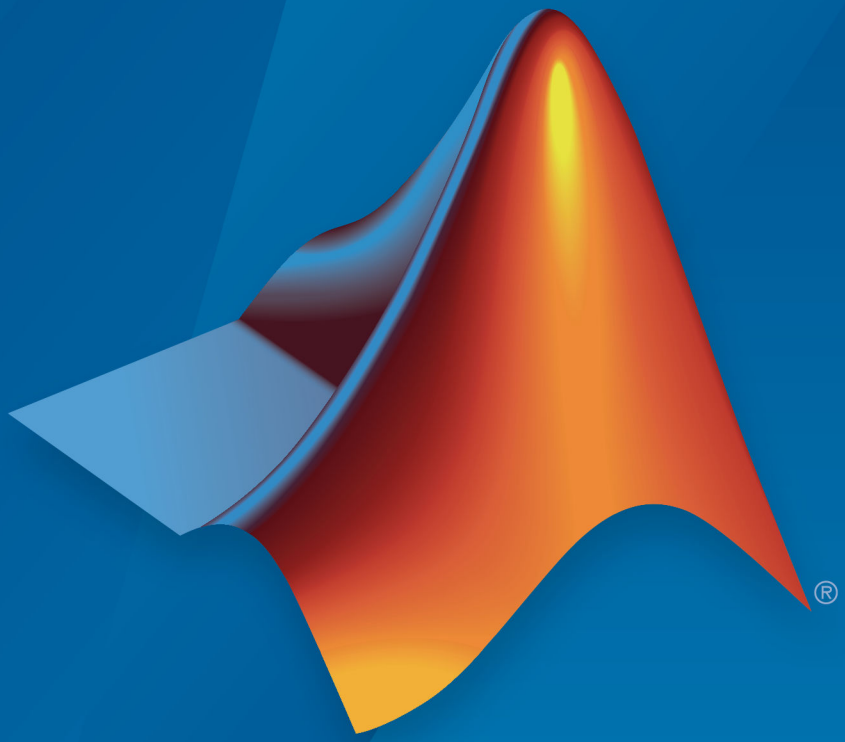
5G Toolbox™

User's Guide



MATLAB®

R2018b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

5G Toolbox™ User Guide

Revision History

September 2018 Online only New for Version 1.0 (Release 2018b)

Contents

1 End-To-End Simulation

2 Signal Reception

3 Code Generation and Deployment

1

End-To-End Simulation

Transmission over MIMO Channel Model with Delay Profile TDL

Display waveform spectrum received through a Tapped Delay Line (TDL) multi-input/multi-output (MIMO) channel model from TR 38.901 Section 7.7.2 using an `nrTDLChannel` System object.

Define the channel configuration structure using an `nrTDLChannel` System object. Use delay profile TDL-C from TR 38.901 Section 7.7.2, a delay spread of 300 ns, and UT velocity of 30 km/h:

```
v = 30.0;                    % UT velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UT max Doppler frequency in Hz

tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
tdl.DelaySpread = 300e-9;
tdl.MaximumDopplerShift = fd;
```

Create a random waveform of 1 subframe duration with 1 antenna.

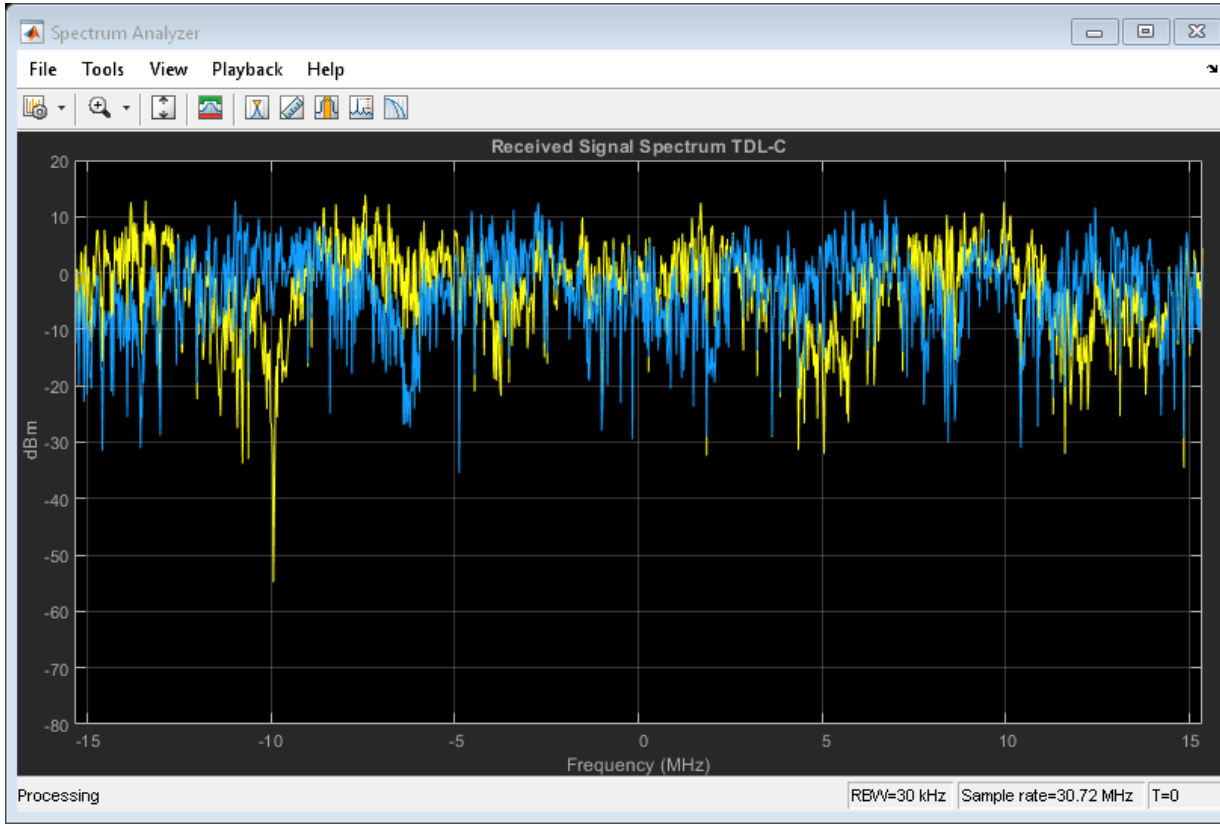
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = tdl(txWaveform);
```

Plot the received waveform spectrum.

```
analyzer = dsp.SpectrumAnalyzer('SampleRate',tdl.SampleRate);
analyzer.Title = ['Received Signal Spectrum ' tdl.DelayProfile];
analyzer(rxWaveform);
```



See Also

System Objects
`nrTDLChannel`

Plot Path Gains for TDL-E Delay Profile with SISO

Plot the path gains of a Tapped Delay Line (TDL) single-input/single-output (SISO) channel using an `nrTDLChannel` System object.

Configure a channel with delay profile TDL-E from TR 38.901 Section 7.7.2. Set the maximum Doppler shift to 70 Hz and enable path gain output

```
tdl = nrTDLChannel;
tdl.SampleRate = 500e3;
tdl.MaximumDopplerShift = 70;
tdl.DelayProfile = 'TDL-E';
```

Configure transmit and receive antenna arrays for SISO operation.

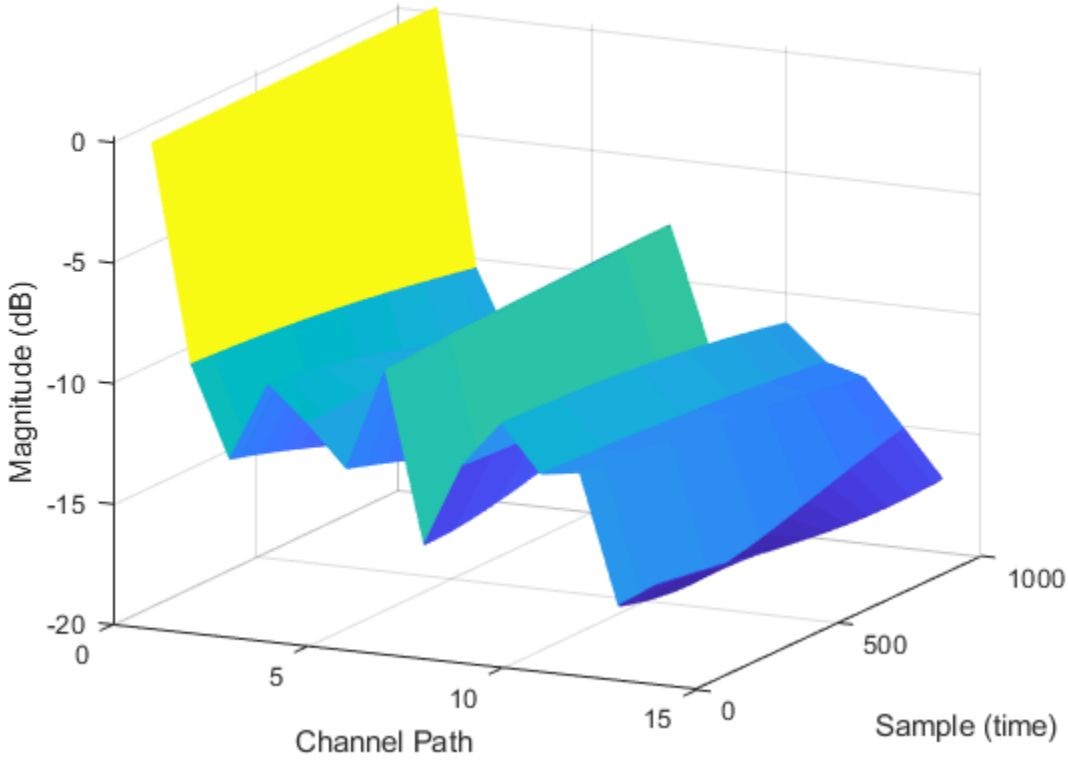
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

Create a dummy input signal. The length of the input determines the time samples of the generated path gain.

```
in = zeros(1000,tdl.NumTransmitAntennas);
```

To generate the path gains, call the channel on the input. Plot the results.

```
[~, pathGains] = tdl(in);
mesh(10*log10(abs(pathGains)));
view(26,17); xlabel('Channel Path');
ylabel('Sample (time)'); zlabel('Magnitude (dB)');
```



See Also

System Objects
`nrTDLChannel`

Reconstruct Channel Impulse Response Using CDL Channel Path Filters

Reconstruct the channel impulse response and perform timing offset estimation using path filters of a Clustered Delay Line (CDL) channel model with delay profile CDL-D from TR 38.901 Section 7.7.1.

Define the channel configuration structure using an `nrCDLChannel` System object. Use delay profile CDL-D, a delay spread of 10 ns, and UT velocity of 15 km/h:

```
v = 15.0;                    % UT velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UT max Doppler frequency in Hz

cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
cdl.DelaySpread = 10e-9;
cdl.CarrierFrequency = fc;
cdl.MaximumDopplerShift = fd;
```

Configure the transmit array as [M N P Mg Ng] = [2 2 2 1 1], representing 1 panel (Mg=1, Ng=1) with a 2-by-2 antenna array (M=2, N=2) and P=2 polarization angles. Configure the receive antenna array as [M N P Mg Ng] = [1 1 2 1 1], representing a single pair of cross-polarized co-located antennas.

```
cdl.TransmitAntennaArray.Size = [2 2 2 1 1];
cdl.ReceiveAntennaArray.Size = [1 1 2 1 1];
```

Create a random waveform of 1 subframe duration with 8 antennas.

```
SR = 15.36e6;
T = SR * 1e-3;
cdl.SampleRate = SR;
cdlinfo = info(cdl);
Nt = cdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
[rxWaveform,pathGains] = cdl(txWaveform);
```

Obtain the path filters used in channel filtering.

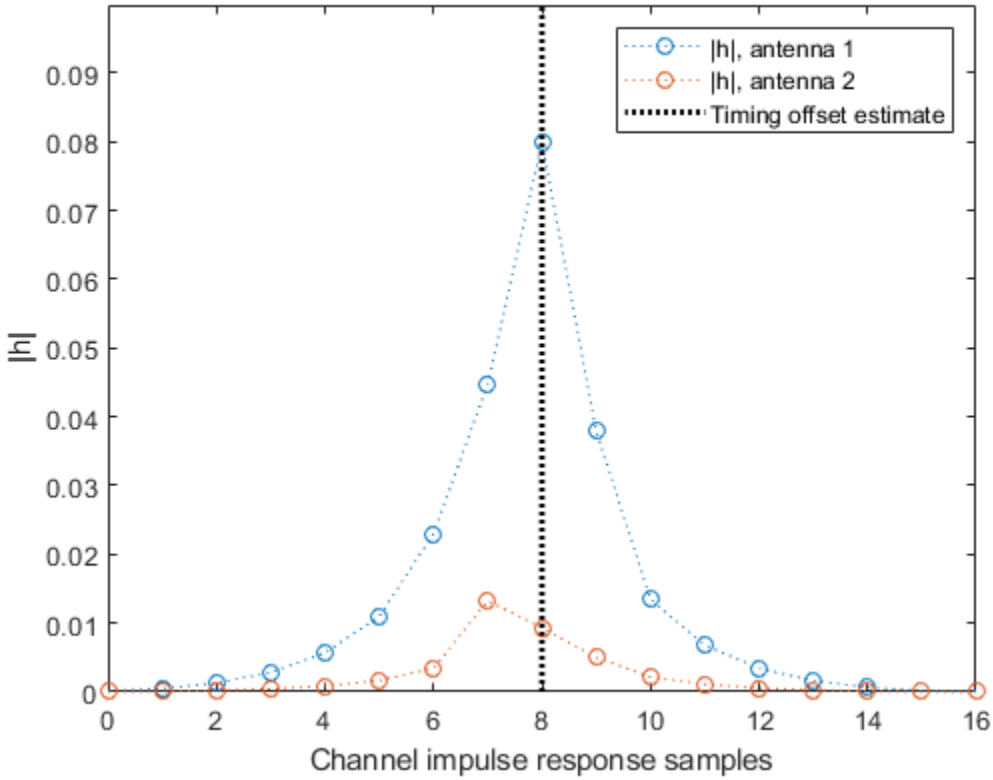
```
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using `nrPerfectTimingEstmate`.

```
[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
```

Plot the magnitude of the channel impulse response.

```
[Nh,Nr] = size(mag);
plot(0:(Nh-1),mag,'o:');
hold on;
plot([offset offset],[0 max(mag(:))*1.25],'k:','LineWidth',2);
axis([0 Nh-1 0 max(mag(:))*1.25]);
legends = "|h|, antenna " + num2cell(1:Nr);
legend([legends "Timing offset estimate"]);
ylabel('|h|');
xlabel('Channel impulse response samples');
```



See Also

Functions

`nrPerfectTimingEstimate`

2

Signal Reception

Extract PBCH Symbols and Channel Estimates for Decoding

Extract physical broadcast channel (PBCH) symbols from a received grid and associated channel estimates in preparation for decoding a beamformed PBCH.

PBCH Coding and Beamforming

Create a random sequence of binary values corresponding to a BCH codeword. The length of the codeword is 864, as specified in TS 38.212 Section 7.1.5. Using the codeword, create symbols and indices for a PBCH transmission. Specify the physical layer cell identity number.

```
E = 864;
cw = randi([0 1],E,1);
ncellid = 17;
v = 0;
pbchTxSym = nrPBCH(cw,ncellid,v);
pbchInd = nrPBCHIndices(ncellid);
```

Use `nrExtractResources` to create indices for the two transmit antennas of a beamformed PBCH. Use these indices to map the beamformed PBCH into the transmitter resource array.

```
P = 2;
txGrid = zeros([240 4 P]);
F = [1 1i];
[~,bfInd] = nrExtractResources(pbchInd,txGrid);
txGrid(bfInd) = pbchTxSym*F;
```

OFDM modulate the PBCH symbols mapped into the transmitter resource array.

```
txWaveform = ofdmmod(txGrid,256,[22 18 18 18],[1:8 249:256].');
```

PBCH Transmission and Decoding

Create and apply a channel matrix to the waveform. Receive the transmitted waveforms.

```
R = 3;
H = dftmtx(max([P R]));
H = H(1:P,1:R);
H = H/norm(H);
rxWaveform = txWaveform*H;
```

Create channel estimates including beamforming.

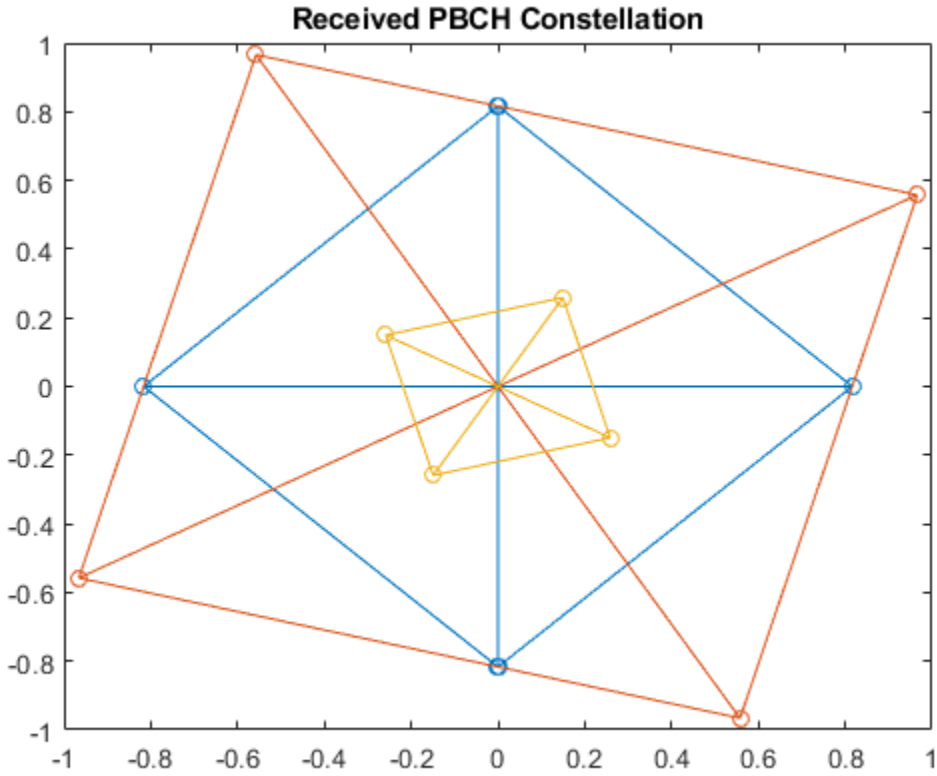
```
hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
nEst = 0;
```

Demodulate the received waveform using orthogonal frequency division multiplexing (OFDM).

```
rxGrid = ofdmdemod(rxWaveform,256,[22 18 18 18],0,[1:8 249:256].');
```

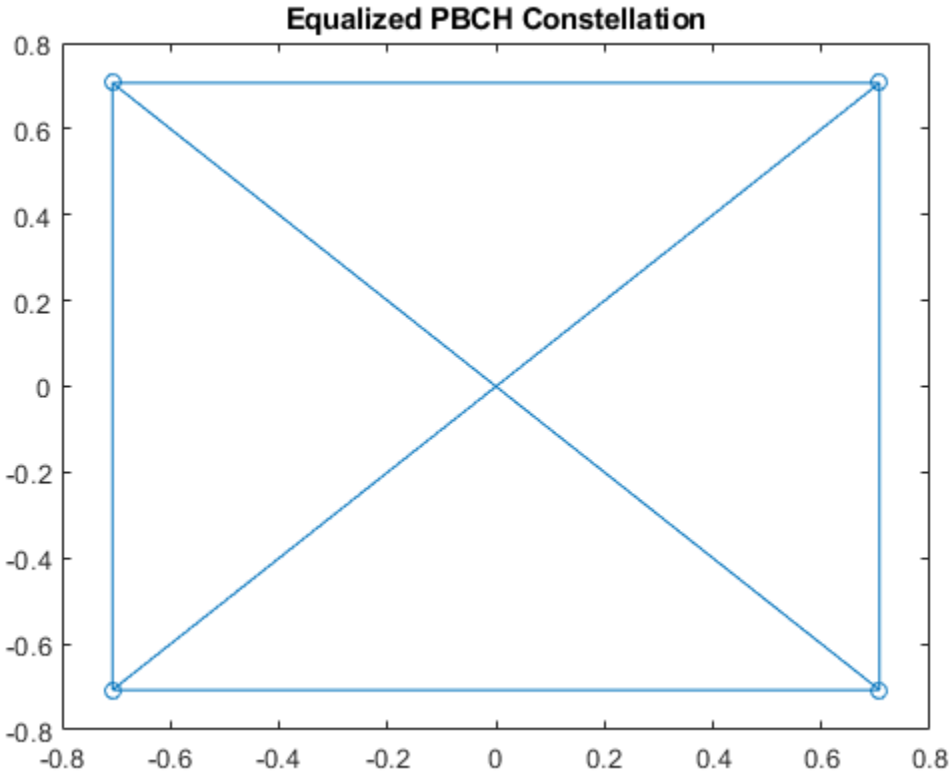
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[pbchRxSym,pbchHestSym] = nrExtractResources(pbchInd,rxGrid,hEstGrid);
figure;
plot(pbchRxSym,'o:');
title('Received PBCH Constellation');
```



Equalize the symbols by performing MMSE equalization on the extracted resources. Plot the results.

```
pbchEqSym = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
figure;
plot(pbchEqSym,'o:');
title('Equalized PBCH Constellation');
```



Retrieve softbits by performing PBCH decoding on the equalized symbols.

```
pbchBits = nrPBCHDecode(pbchEqSym,ncellid,v)
```

```
pbchBits = 864×1
10^10 ×

   -2.0000
   -2.0000
    2.0000
   -2.0000
   -2.0000
    2.0000
    2.0000
```

```
   -2.0000
   -2.0000
   -2.0000
      ⋮
```

See Also

Functions
nrEqualizeMMSE | nrExtractResources

Code Generation and Deployment

What is C Code Generation from MATLAB?

You can use 5G Toolbox together with MATLAB® Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see "Getting Started with MATLAB Coder" (MATLAB Coder).

Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the `"mex"` suffix is added.

```
codegen myfunction.m
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development

Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “C/C++ Code Generation” (MATLAB Coder) for more information.

C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read “Setting Up the C or C++ Compiler” (MATLAB Coder). After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

All 5G Toolbox functions and System objects support for code generation. For an overview, see “Functions and System Objects Supported for MATLAB Coder” on page 3-4.

See Also

Functions
`codegen` | `mex`

More About

- “MATLAB Code for Code Generation Workflow Overview” (MATLAB Coder)
- “Functions and System Objects Supported for MATLAB Coder” on page 3-4

Functions and System Objects Supported for MATLAB Coder

You can generate efficient C/C++ code for all 5G Toolbox functions and System objects by using the MATLAB Coder product (requires a license).

An asterisk (*) indicates that the reference page has usage notes and limitations for C/C++ code generation.

getPathFilters
info
nrBCH
nrBCHDecode
nrCDLChannel*
nrCodeBlockDesegmentLDPC
nrCodeBlockSegmentLDPC
nrCRCDecode
nrCRCEncode
nrDCIDecode
nrDCIEncode
nrDLSCHInfo
nrEqualizeMMSE
nrExtractResources
nrLayerDemap
nrLayerMap
nrLDPCDecode
nrLDPCEncode
nrPBCH
nrPBCHDecode
nrPBCHDMRS
nrPBCHDMRSIndices

nrPBCHIndices
nrPBCHPRBS
nrPDCCH
nrPDCCHDecode
nrPDCCHPRBS
nrPDSCH
nrPDSCHDecode
nrPDSCHPRBS
nrPerfectChannelEstimate
nrPerfectTimingEstimate
nrPolarDecode
nrPolarEncode
nrPRBS
nrPSS
nrPSSIndices
nrRateMatchLDPC
nrRateMatchPolar
nrRateRecoverLDPC
nrRateRecoverPolar
nrSSS
nrSSSIndices
nrSymbolDemodulate
nrSymbolModulate
nrTDLChannel*

See Also

More About

- “What is C Code Generation from MATLAB?” on page 3-2